Spring 5-1-2005

Software Version Control Tools with Implementation

Zhu Wang
Dakota State University

Software Version Control Tools with Implementation

Zhu Wang

A project submitted in partial fulfillment of the requirements for the

Master of Science in Information Systems

Dakota State University

2005

DAKOTA STATE dsu UNIVERSITY

MSIS
PROJECT APPROVAL FORM

Student Name: Zhu Wang

Expected Graduation Date: May 7, 2005

Master's Project Title: Software Version Control Tools with Implementation

Date Project Plan Approved: April 28

Date Project Coordinator Notified and Grade Submitted: April 28

Approvals/Signatures:

Student: Zhu Wang Date: 4/28/05

Faculty supervisor: [signature] Date: 4/28/05

Committee member: [signature] Date: 4/28/05

Committee member: [signature] Date: 4/28/05

To Ensure Certification of Completion:
Student must bring or send the original to the Graduate Programs Office.
Copies on acid free paper go to the library with the reports for binding.

Original to Graduate Programs Office
Acid-free copies with written reports to library
Copies to: Project supervisor and committee and to MSIS Coordinator

Revised 1/28/03

Abstract

Version control is an important part of Software Configuration Management. In this project, properties of version control tools have been analyzed, and *Simple Version Control System* (SVCS) application with common version tool functions is developed. SVCS is built in Java to achieve high portability and to decrease design difficulties in developing distributed applications. SVCS' architecture, package design, class design, and implementation are discussed in the project report. SVCS is successful in architecture and common functions. Performance and functionality need to be improved in the future development of SVCS.

Table of Contents

List of Figures

1. Introduction

During software project development, every unit of the project may undergo changes. The changes can be found in project documents, source codes, releases, development processes or resources. A mature development process can establish and maintain the integrity of the artifacts of the software project throughout the project's software life cycle, which is the purpose of Software Configuration Management (SCM) [1].

SCM is a process to identify configuration items, control their changes during the project development, and it has the ability to store and recover the configuration item/unit at the given time. After more than 20 years of research and practice, SCM has become the most important part of project management. It has been addressed in all kinds of popular quality/process models, such as ISO-9000, SEI-CMM, ISO-12207, MIL-498, RUP, and etc. And the scope of SCM has become more and more abroad [2]. The contents of SCM include version control, change control, baseline control, build control, release control, and etc.

Version control is the core function of SCM, and it provides the repository for the storage of configuration items and their changes. The earliest SCM tools actually only implemented version control. Version control will focus on the changes of visible files, which may be source codes, binary files, and documents. Whenever bugs are fixed, new functions are implemented, or original functions are enhanced, new versions are introduced to record changes to one or more related files. Version control provides the process and tools to maintain the consistency and integrity of versions. Version control is

often accompanied with or mixed with Change Control, which will authorize, review and audit the changes to the archived versions.

Version control has several important capabilities:

- It allocates the location(s), centralized or distributed, for a group of people to store and share files.
- It places controls on who can see, read, and alter files.
- It defines methods to compare the difference of two different versions.
- It stores a complete history of all changes made to all files.
- It can recover a given version of a file.

Software projects are always full of conflicts and confusions if the developing processes are not well defined. The following typical chaos can be resolved by version control,

- Customer reports a bug, but you are not sure which version is the exact source introduced this error.
- Two programmers try to modify the same piece of source code. Whose result should be accepted? Or can their jobs be merged without conflicts?
- After two months' development, you sadly decide to give up and go back to the old status, and you suddenly find that there is no way to retrieve old versions, as the original source codes have been replaced with recent junks.

- You fixed a serious security hole in the current release, and you wonder how to migrate the same enhance to all previous releases.

Today, large scale of team, parallel and distributed development, and multiple releases make the situation even worse if there's no version control during development.

Version control is originated from UNIX system, SCCS [4], RCS [5] are among the earliest version control systems. Currently, there are many mature commercial or free version control tools, including Rational ClearCase, Visual Source Safe (VSS), PVCS, CVS and Subversion. CVS now is the most popular version control system in Open-Source Community. Subversion is based on CVS, and a better system than CVS, however it doesn't gain the same popularity as CVS. On Windows platforms, there're fewer choices. Many windows application programmers are only familiar with VSS.

2. Statement of problem

2.1. Version, Revision and Variant

There is a tremendous amount of overloading of terminology in the field of SCM. In order to have a clear description of the project and avoid confusion, we'll use the following definitions [2]:

Variants of configuration items are different implementations that remain valid at a given instant in time, created to handle environmental differences (for example, different execution platforms).

Revisions are the steps a configuration item goes through over time, whether to handle new features, fix bugs or to support permanent changes to the environment (e.g., operating systems upgrades, if the old one is no longer supported).

Variants and revisions provide a two-dimensional view into the repository, with variants incrementing along one axis as required and revisions incrementing through time on the other.

Versions of configuration items are understood by the SCM community to be synonymous with either revisions or variants [6]. Therefore a version of a single configuration item denotes an entry in the two-dimensional view of the repository reached from an origin through some path of revisions and variants. In most case, the meaning of *version* is clear within the context.

2.2. Common Version Control Functions

Version control system has become more and more powerful. Here are the most common functions of a version control system,

2.2.1. Check-in

Check-in adds a new file or a revision to the repository, which means check-in has write permission. The system should define who has the write permission to check in. New file creator and revision owner of course can check-in. A check-in may be aborted if there's no modification.

2.2.2. Check-out

Check-out retrieves a revision from repository. The revision checked out can be read-only or writeable. Read-only revision can be checked out without restriction, which means that the same revision can be checked out by multiple different users. To check out a writeable revision is implemented differently in different version control systems because of the difference in lock schemes. One simple scheme is that at most one user keeps a writeable revision at a given time, so this type lock scheme can avoid the confliction that could be brought about if several users change the same revision concurrently.

2.2.3. Diff

Diff calculates the difference between two versions. Repository uses diff to help organize the storage structure of different versions. Some systems only deal with the differences

between text files, while recent systems can show the differences between binary files. The output of differences can be displayed in console or in visual user interface.

2.2.4. Branch

Branch creates a variant of file. Suppose you are going to transit the current software project on UNIX to Windows platform, you need not to code from scratch. You can create a variant from current version, and develop on it. The situation is very common in parallel development. The following figure shows the idea of branch.

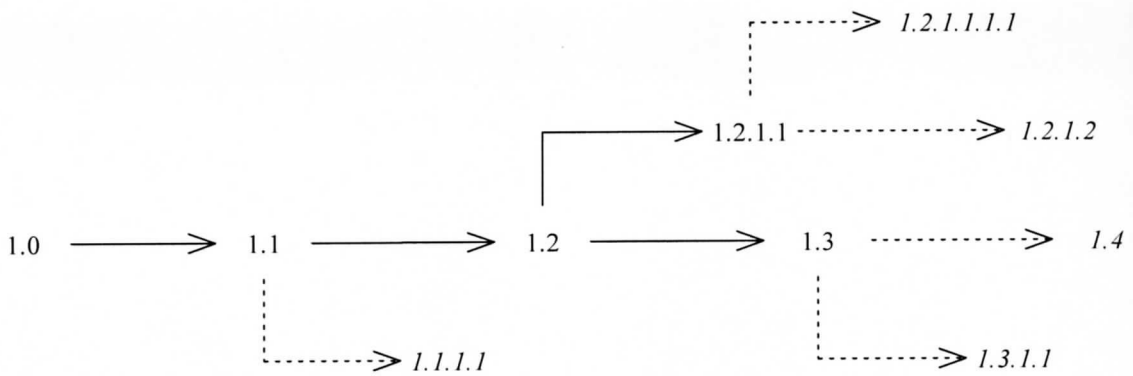

Figure 2.1 An Example of branch

Figure 2.1 looks like a tree. The simple revisions *x.x* are called *trunk* versions, while the set of revisions of the variant is called a *branch* because they look like a branch of the tree. In figure 2.1, there's a branch 1.2.1, and two branch versions, 1.2.1.1 and 1.2.1.2. Trunk version 1.2 is called a branch-point.

2.2.5. Merge

Merge incorporates the changes from multiple revisions of a file into one revision. It's useful in concurrent development environment. However, conflicts and overlapping between different users' revisions should be detected. If conflicts exist, automatic merge is not possible. Some system will create a communication channel between the authors with conflicting revisions.

2.3. SVCS

Most current version control systems are developed with C/C++. The advantage is speed and direct manipulation of the file system. One possible weakness is the portability. One example is end-of-new-line. UNIX treats line feed symbol '\n' as the end of new line, while Windows adds an additional carriage return to it to represent new line. So under Windows, a new line is ended with '\r\n'. If a version control tool hard-coded '\n' or '\r\n' to identify new line, the file created under a different operation system may be treated improperly by the tool.

CVS now is the most popular free version control system. However the server can only run under UNIX. If you want to operate on Windows platform, you need to install its Windows variant CVSNT. CVS client also has the same problem. So there're CVS client for UNIX, client for Windows, and client for Macintosh.

As for the portability, Java should be a better choice. Java application can run on different platforms without re-compilation on most conditions. So you need not to worry whether the server and/or client are on UNX or Windows. This is also the reason why the project came into being. Although there're a lot of version control systems, it's still worth

developing a more portable system with Java. Actually there're only very few of full-featured version control systems written in Java.

This project will develop the *Simple Version Control System* (SVCS) in Java. "*Simple*" means that SVCS is aimed to implement common version control functions. However, based on the well-defined architecture of SVCS, other extensive functions can be easily added to it.

3. Objectives of the project

Simple Version Control System (SVCS) developed in this project should satisfy the following objectives.

3.1. Portability

SVCS should be run on different platforms. In another word, both SVCS server and SVCS clients can be installed and operate on different platforms.

3.2. Compatibility

SVCS should be compatible with RCS/CVS. SVCS will use RCS' file format to store the version history of a file.

3.3. Functionality

SVCS should have the most common version control functions. The functions to be implemented are:

- Check-in
- Check-out
- Set working directory
- Create / Delete Project
- Visual Diff
- Branching

- Refreshing
- Sorting

SVCS doesn't implement merge, which will be realized in the future. Actually there're no technical difficulties to provide this function, but how to make merge become practical is not easy. Communications between developers can never be replaced by automatic tools.

3.4. Deliverables

Deliverables of SVCS project include:

- Source Codes
- Executable Packages
- Installation Files
- Quick Guide Document
- Final Project Document

4. Implementation

4.1. Architecture Overview

Most version control system is centralized, and the system can be divided into Repository and Client. Repository is more like a specific file server which can store and retrieve the files and all the changes happened upon them. The architecture of centralized version control system is a Client/Server model. Clients issue requests to read the existing versions or to write/add a new version of a file by connecting to the repository, while centralized repository responses clients' requests and stores data.

SVCS adopts Client/Server model that is illustrated in figure 4.1.

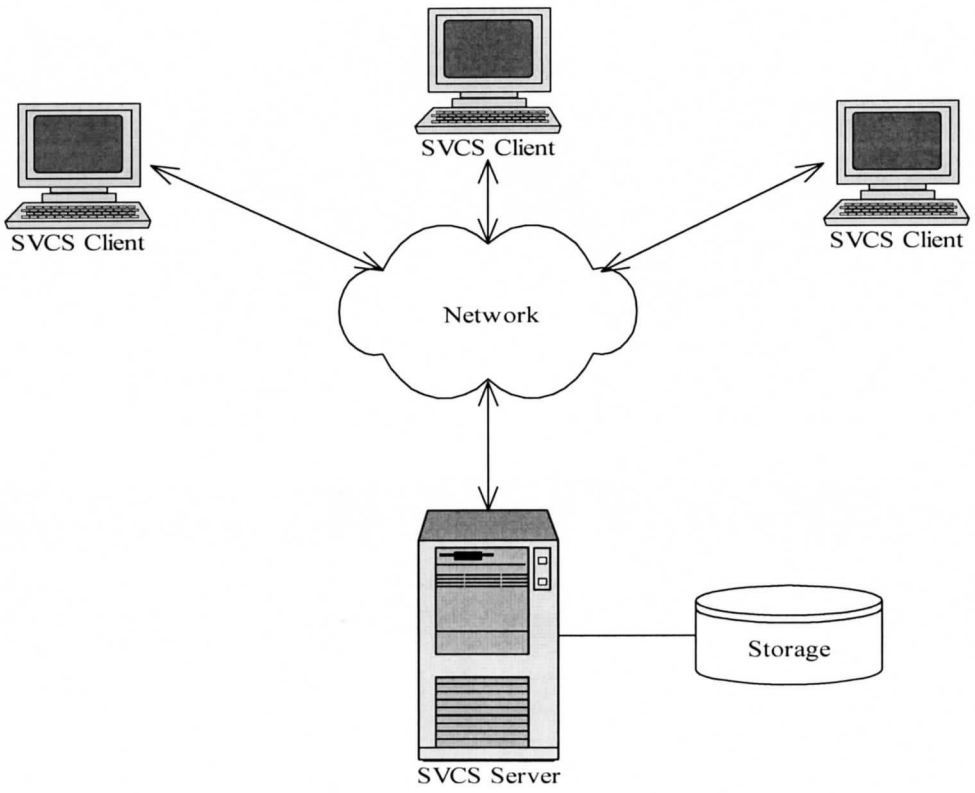

Figure 4.1 Client/Server Architecture of SVCS

4.1.1. Programming Language

Java is the programming language of SVCS. Java's most characteristic feature is that it is a platform-independent language. This means that Java programs will run on any machine that supports the language. The slogan "*write once, run anywhere*" has often been used to describe and promote the Java language (However sometimes it's not the truth).

The second reason to choose Java is its powerful built-in network programming packages. As the so-called "*Internet Programming Language*", Java supports low-level socket programming, distributed programming (CORBA, RMI), naming and directory service (JNDI), and Web Service (XML-based RPC, WSDL). Also there're free packages for P2P programming. Any developers bitten by socket programming should have a close look at Java's distributed and Internet program APIs. SVCS is a C/S application, and Java should be a good choice for the development.

Visual Interface on difference platforms is the third reason. Java provides Swing package for interface design. Swing offers huge graphic components to use, and it can build any sophisticate software interface. "Write once, run anywhere" works here also. Same user interfaces can be achieved on different platforms just coding once. Using native API to write graphic interface for different platforms is not an interesting task.

The biggest concern by using Java is its packages (io and nio package) for file system. Because file system is Operation System dependent, Java has quite a few compatible issues in IO packages. And some important features, including file locking, are not

supported until J2sdk1.4. To simplify the programming, SVCS uses simple preempt locking scheme that ensures at most one user keeps a writeable revision at a given time.

Finally, J2sdk1.4 is selected as the programming tool. The last reason is that SVCS uses Regular Expression (Regex) package, which built into Java SDK until 1.4, to develop file parser.

4.1.2. Distributed Programming

SVCS is a centralized system, and every direct access to the archived versions should be done by the server. If you are an aged programmer, you may think of RPC (Remote Procedure Calls). The client sends request parameters to the server, while the server executes the request, and responses the results back to the client. The request parameters and response results here may be objects, but the old-fashioned RPC can only transfer unstructured data.

In Java, we can choose RMI, CORBA, or XML-based RPC to handle the interactions between client object and server object. These three solutions all have advantages and disadvantages.

The fundamental difference between CORBA and RMI is that CORBA is language neutral. Both server and client part of SVCS are developed in Java, so we need not to use CORBA. In most case, CORBA is more powerful but more complicate. RMI is easier to use but only suitable in Java environment.

RMI let you to access remote object, and call its methods as if the methods are available in local machine. So the client can call any methods the server exposed to the client, and

the client need not worry how to encode and send the request, accept and decode the results, all these jobs are handled by RMI. Of cause, the simplicity is at the cost of performance, because RMI needs to load and transfer the object through the network. XML-based RPC is more efficient than RMI, and there's one version control system used this solution.

SVCS uses RMI mostly because of its simplicity. By using of RMI, the design and deployment of SVCS become much easier.

4.1.3. Delta Algorithm

Delta algorithms are used to compute difference between two files. The difference is called *Delta*. The early applications of delta algorithms occurred in version control systems such as SCCS and RCS. During project development, the size of the difference between two consecutive revisions usually is much less than the size of old revision. So, by storing deltas relative to a base revision, repository can save substantial amounts of disk space compared with storing every revision file.

Delta compression not only saves space, but also reduces I/O and network traffics, so the performance of version control systems can be improved.

The classic program for generating deltas is Unix *diff* [7; 8]. Both SCCS and RCS use it for storage and display of differences. *diff* is limited to text files, and now there are many delta algorithms also can handle binary files, such as word processor files, spreadsheet data, electrical and mechanical CAD data, audio, and images.

During the design of repository, two important issues are related to delta algorithm,

- How to compute the delta between two revisions?

- How to store and retrieve revision history?

The first issue is to obtain a good algorithm. The second issue is how to apply the algorithm. Here is a simple example. Supposing the original revision *Ver1.cpp* is

```
#include <iostream>
using namespace std;
void main(){
        cout<<"Hello"<<endl;
        cout<<"Hello Word"<<endl;
        cout<<"Hello C++ World"<<endl;
}
```

The content of new revision *Ver2.cpp* is,

```
#include <iostream>
using namespace std;
void main(){
        cout<<"Hello World"<<endl;
        cout<<"Hello C++ World"<<endl;
        cout<<"Hello"<<endl;

}
```

If we calculate delta *D* based on *Ver1.cpp* (the old revision), *D* is called *Forward Delta*. And *Ver2.cpp* can be reconstructed from *Ver1.cpp* by applying delta *D* to the latter.

Unix diff shows the result as

```
% diff  Ver1.cpp  Ver2.cpp
4,5c4
<       cout<<"Hello"<<endl;
<       cout<<"Hello Word"<<endl;
---
>       cout<<"Hello World"<<endl;
6a6
>       cout<<"Hello"<<endl;
```

Here the output means line 4 through line 5 of *Ver1.cpp* are changed to line 4 of *Ver2.cpp*, and after line 6 of *Ver1.cpp* append a new line to form line 6 of *Ver2.cpp*.

The repository doesn't store the delta in the above format, while it stores *Edit Scripts.* In this example the edit scripts may be:

```
d4 2
a5 1
        cout<<"Hello World"<<endl;
a6 1
        cout<<"Hello"<<endl;
```

Associated with *Ver1.cpp*, there are three edit scripts: d4 2 is to delete 2 lines from line 4, a5 1 is to append a line after line 5, and a6 1 is to append a line after line 6. So you can see that *Change* is equal to *Delete* combined with *Append.* Executing these scripts to *Ver1.cpp*, we can get *Ver2.cpp.*

If we calculate delta *D* based on *Ver2.cpp (the new revision), D* is called *Reverse Delta.* And *Ver1.cpp* can be reconstructed from *Ver2.cpp* by applying delta *D.*

The Edit Scripts stored with *Ver2.cpp* in the repository becomes:

```
d4 2
a6 2
        cout<<"Hello Word"<<endl;
        cout<<"Hello C++ World"<<end;
```

Executing these scripts to *Ver2.cpp*, we can get *Ver1.cpp.*

SVCS uses the algorithm proposed by Myers [3]. This is a simple O(*ND*) time and space algorithm is developed where *N* is the sum of the lengths of file *A* and file *B* and *D* is the size of the minimum edit script for *A* and *B*. The algorithm performs well when differences are small (sequences are similar) and is consequently fast in typical applications.

Because the access of latest revision should be more frequent than that of early revisions, SVCS uses reverse delta to store revision history. So old revisions need to be reconstructed from the latest revision, and content of latest revision is always kept.

4.1.4. File Format

SVCS' file format is compatible with that of RCS/CVS. Here is the formal description of SVCS file content.

```
SVCSFile    ::= admin
                {delta}*
                desc
                {deltatext}*
admin       ::= head {ver};
                { branch   {ver}; }
                access {id}*;
                symbols {sym : ver}*;
                locks {id : ver}*;  {strict  ;}
                { comment  {string}; }
                { expand   {string}; }
                { newphrase }*
delta       ::= ver
                date ver;
                author id;
                state {id};
                branches {ver}*;
                next {ver};
                { newphrase }*
desc        ::= desc string
deltatext   ::= ver
                log string
                { newphrase }*
                text string
ver         ::= {digit | .}+
digit       ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
id          ::= {ver} idchar {idchar | ver }*
sym         ::= {digit}* idchar {idchar | digit }*
idchar      ::= any visible graphic character except special
special     ::= $ | , | . | : | ; | @
string      ::= @{any character, with @ doubled}*@
newphrase   ::= id word* ;
word        ::= id | ver | string | :
```

Symbol "|" separates alternatives; "{}" enclose optional phrases; "*" means that the preceding phrase can be repeated zero or more times; "+" means that the preceding phrase must appear once and can be repeated more times. Here are some more explains:

- *Head* version means the latest version.
- *Branch* version means if the default branch has been set. If so, check-in or check-out will be on branch, not on trunk.
- *Access* list contains the users who are allowed to make modifications to the SVCS file.
- *Locks* is a list of revisions locked by different users.
- *Date* has a format of "yyyy.MM.dd.HH.mm.ss".
- If *String* contains "@", it should be doubled.
- *Text* of latest version is the revision's content, and *Text* of previous version is the edit scripts.

The following list is a sample *admin* part of SVCS file:

```
head    1.3;
access;
symbols;
locks
        wangz:1.1
        Administrator:1.2; strict;
comment         @This is an example. @;
```

The following list is a sample *delta* part of SVCS file:

```
1.2
date    2003.08.20.05.43.44;  author Administrator;  state Test;
```

```
branches
        1.2.1.1;
next    1.1;
```

The following list is a sample *deltatext* part of SVCS file:

```
1.1
log
@Initial revision
@
text
@line1d1 2
a2 1
line1@
```

4.2. Class Design

4.2.1. Package Diagram

SVCS can be divided into client subsystem and server subsystem, and the package diagram is illustrated in figure 4.2.

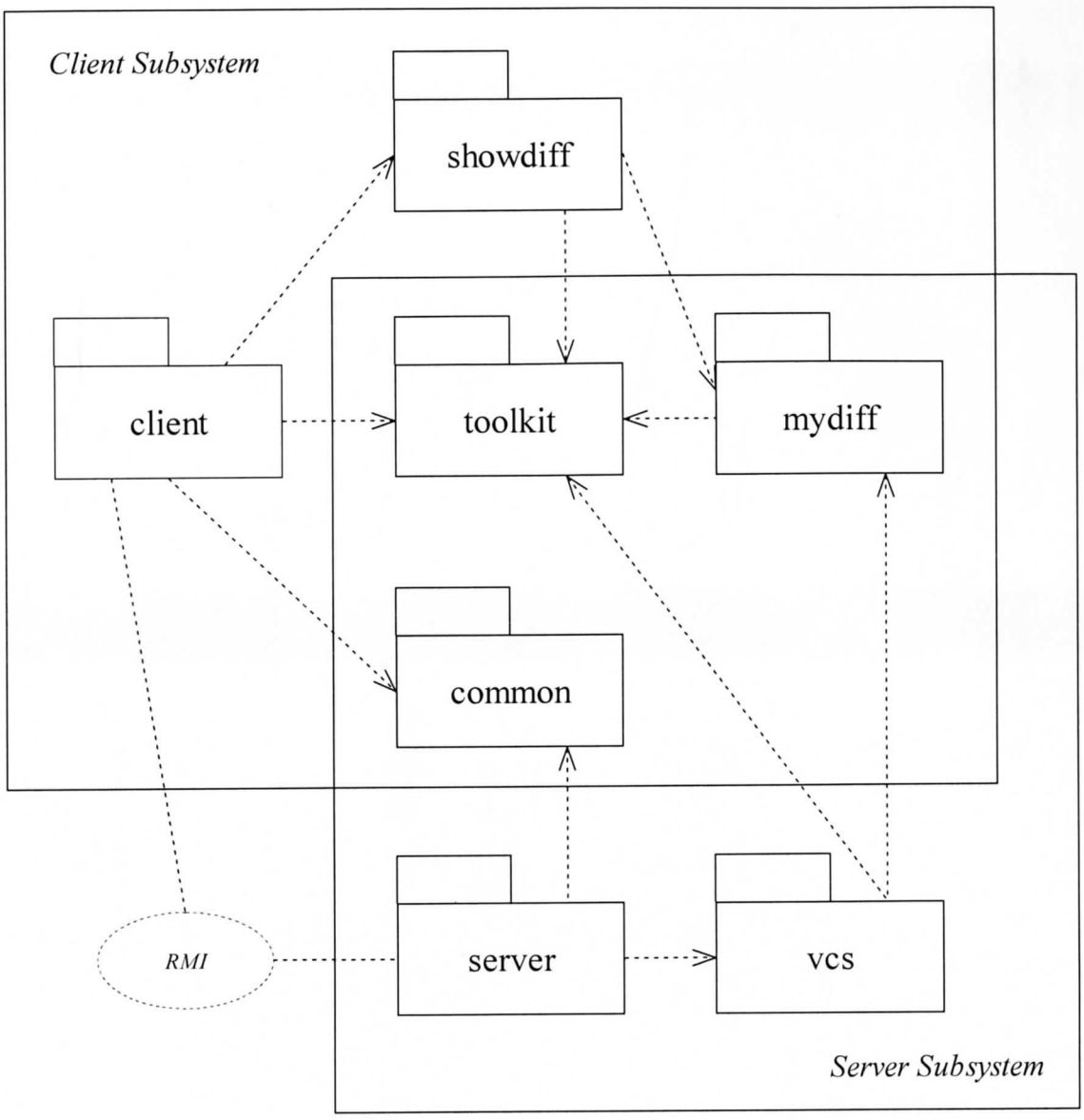

Figure 4.2 Package Diagram of SVCS

Client subsystem includes client, showdiff, mydiff, toolkit, and common packages. Server subsystem includes server, vcs, mydiff, toolkit, and common packages. Toolkit, common and mydiff packages are included in both client and server end. Client and server are interacted through RMI.

4.2.2. toolkit Package

Package toolkit contains only one class Tools, which provide several useful static functions.

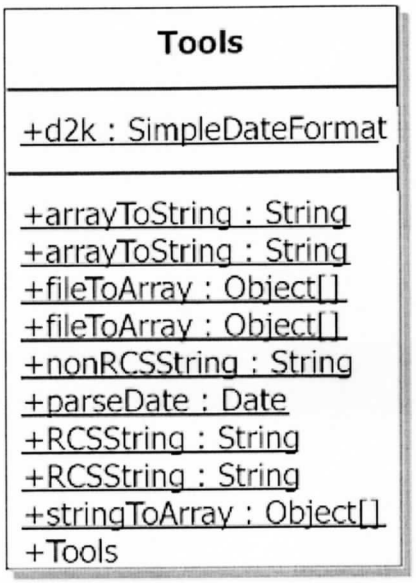

Figure 4.3 Tools class

- Operation fileToArray and stringToArray convert a file or a piece of string into an array, and each line will be one element.
- Operation arrayToString converts an array into a string.
- Operation RCSString and nonRCSString encode and decode a *Delta Text.*
- Operation parseDate parses a string into an object of *Gregorian* calendar.
- Property d2k represents defines the display format of "yyyy-MM-dd HH:mm:ss" for a date.

4.2.3. common Package

Package common defines four types of persistent object: RCSProject, TreeNodeObject, TableRowObject and RCSFileInfo. In java a persistent object is created by realizing java.io.Serializable interface. SVCS uses them to transfer information through network, and to store information permanently by writing it to disk.

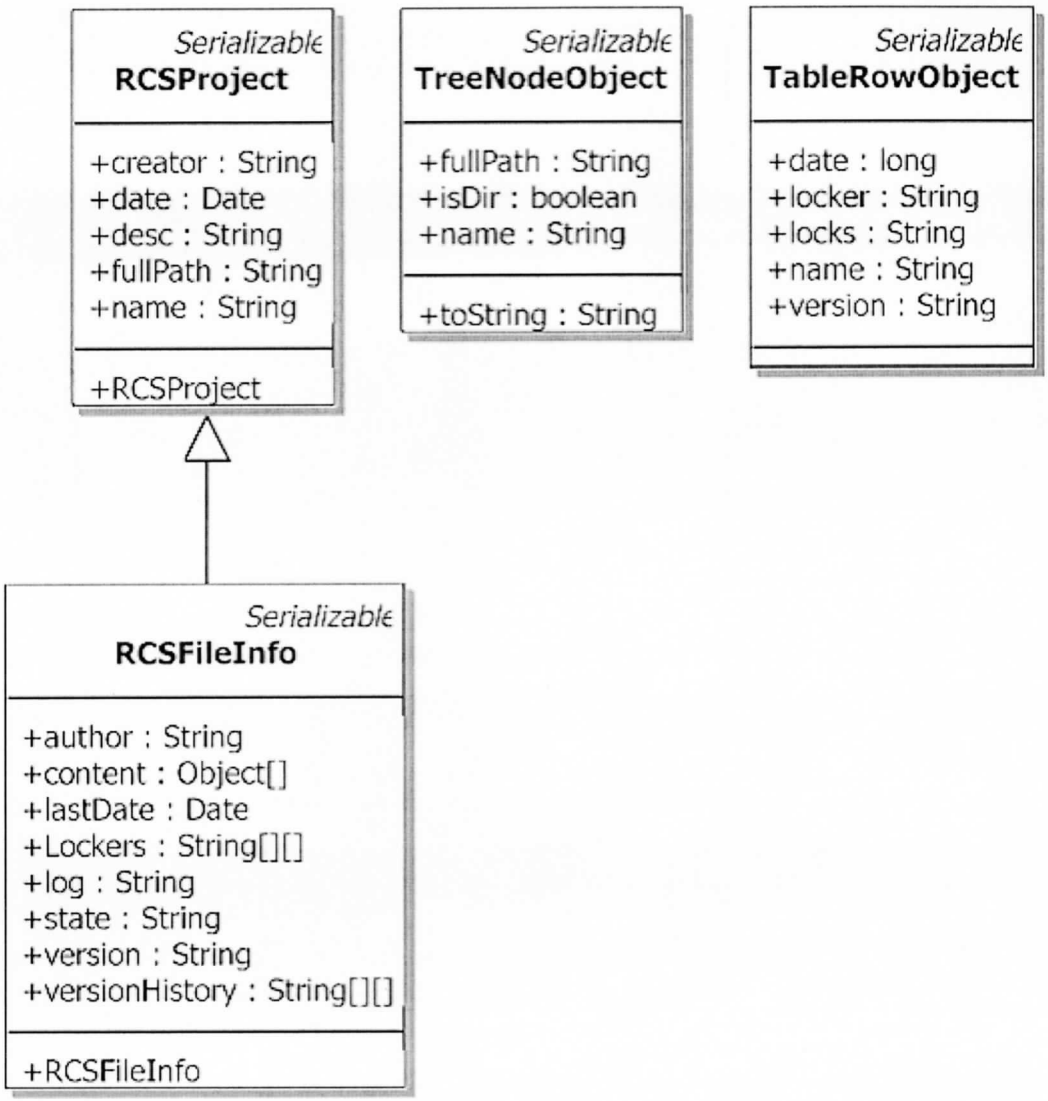

Figure 4.4 Class Diagram of Package common

- RCSProject contains information of projects, such as project name, creator, created date, and description.
- RCSFileInfo contains content of a RCS-format file.
- TreeNodeObject is used to determine if a file in repository is a directory or not. Directory and leaf file will be displayed with different icons on client's user interface.
- TableRowObject contains five fields that will be displayed as five columns of a spreadsheet.

4.2.4. mydiff Package

Package mydiff has one core class Diff, two structures EditScript and PathPoint, and two containers EditScripts and PathPoints. Diff class implements the delta algorithms proposed by Eugene W. Myers.

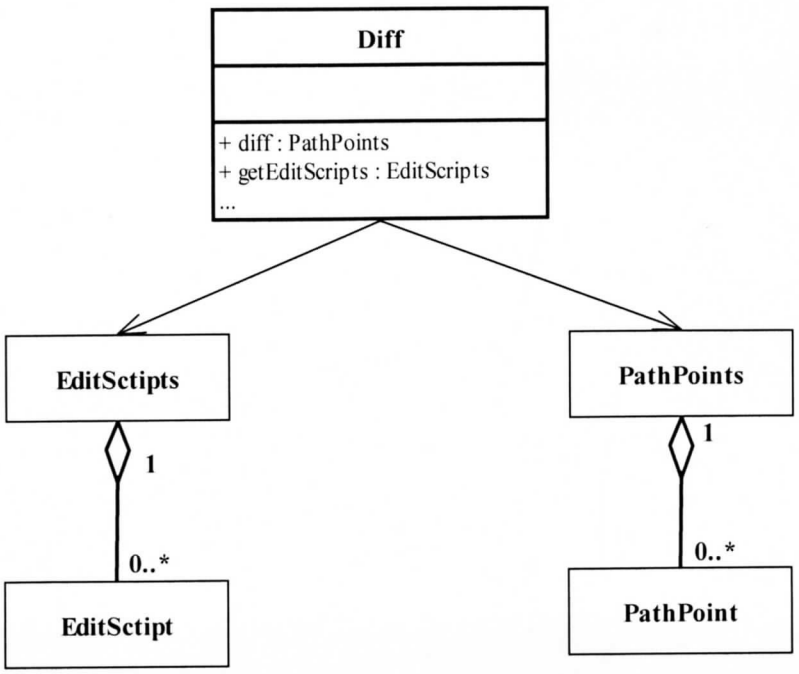

Figure 4.5 Class Diagram of Package mydiff

- Diff class calculates the delta between original file and new file. The LCS (longest common sequence) is stored in PathPoints. And PathPoints can be translated into EditScripts.

- PathPoints class is extended from java.util.Vector, and it's responsible for holding PathPoint objects.

- PathPoint depicts the pattern of a *non-diagonal edge* followed by as many as possible of *diagonal edges*. The details can be referred to [3]

- EditScripts class is similar to PathPoints. It is also extended from java.util.Vector , and it's responsible for holding EditScript objects.

- EditScript stores an *edit script* which contains starting positions and ending positions of both original file and new file, and one of the three possible edit script type: *Append*, *Delete*, and *Change*.

4.2.5. showdiff Package

Package showdiff creates two visual components, ShowDiffDialog and NoWrapJTextPane, which can display the difference between two files.

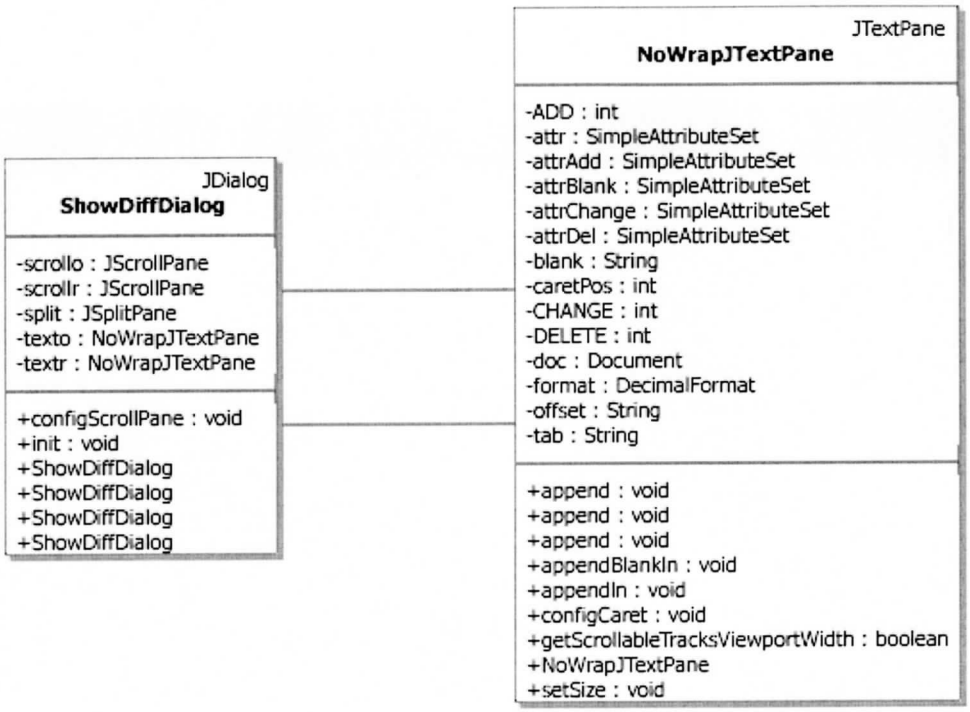

Figure 4.6 Class Diagram of Package showdiff

- NoWrapJTextPane is extended from JTextPane to display the content of a file, and the lines will not be wrapped. So lines of two different files can be aligned neatly.

- ShowDiffDialog is a modeless dialog used to display the difference between two files. Each file is displayed in a NoWrapJTextPane.

4.2.6. client Package

Package client creates a client application to interact with server application and finish SVCS version control functions. All the implementations of the version control functions are defined in server package, so client package just provide a user interface to call these functions. The package uses Swing extensively.

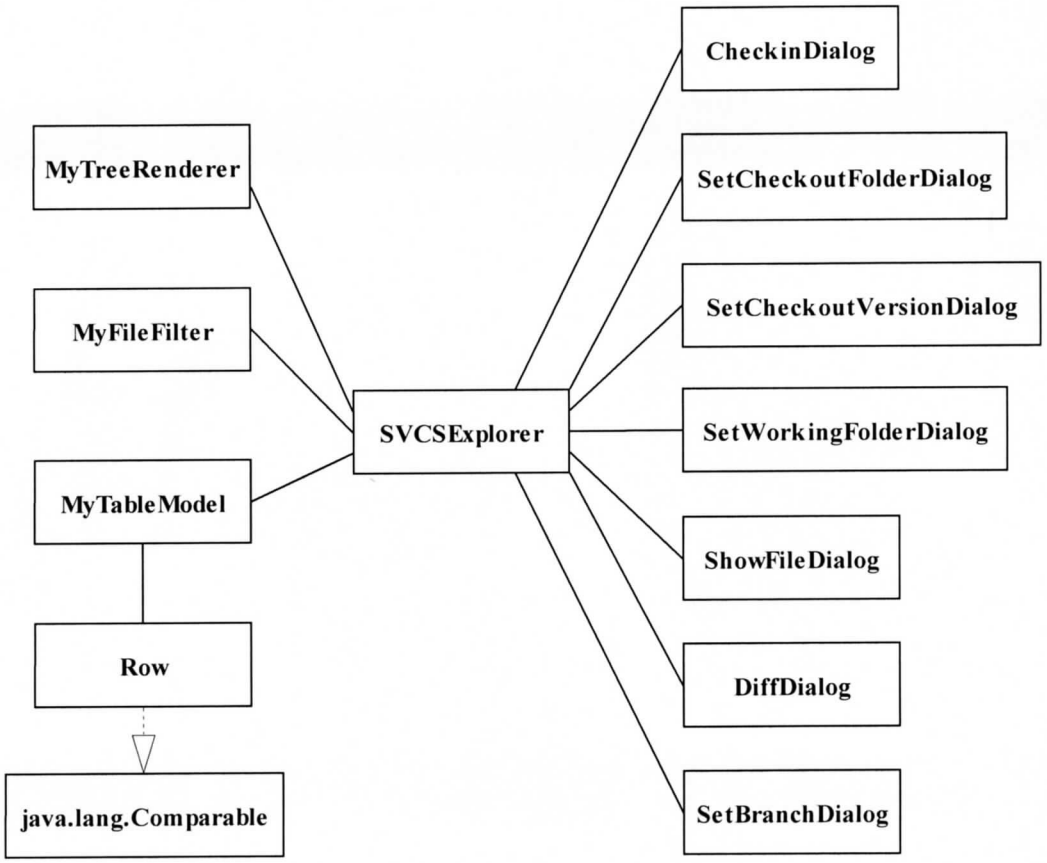

Figure 4.7 Class Diagram of Package client

There are several customized dialog classes in this package. Each dialog is corresponding to a specific version control function, such as check in, check out, diff, set branch, and etc.

The main user interface consists of a JTree component and a JTable component.

- MyTreeRenderer defines how to render the JTree object.
- MyFileFilter filters the files by expected file extension.
- MyTableModel stores the data for the JTable object.
- Row is used to sort the rows in a JTable on a specific column. It implements Comparable interface, so it defines how to compare two rows.

4.2.7. vcs Package

Package vcs defines the file format of repository and the corresponding file parser to retrieve the contents. SVCS uses the RCS compatible file format that is described in **section 4.1.4**.

- Version is a dot-separated digital string. The version with one dot or less is trunk version. The version with even number of dots is called branch. And any other versions with odd number of dots are called branch versions.
- Admin stores admin part of SVCS file format.
- Delta stores delta and deltatext of SVCS file format.
- RCSFile defines SVCS file format. The prefix RCS- means RCS compatible. So RCSFile contains an Admin object, a Delta object, and a description string.
- RCSFileParser parses its seq String attribute into a RCSFile or a part of RCSFile. Attribute seq is initialized during construction of RCSFileParser object.

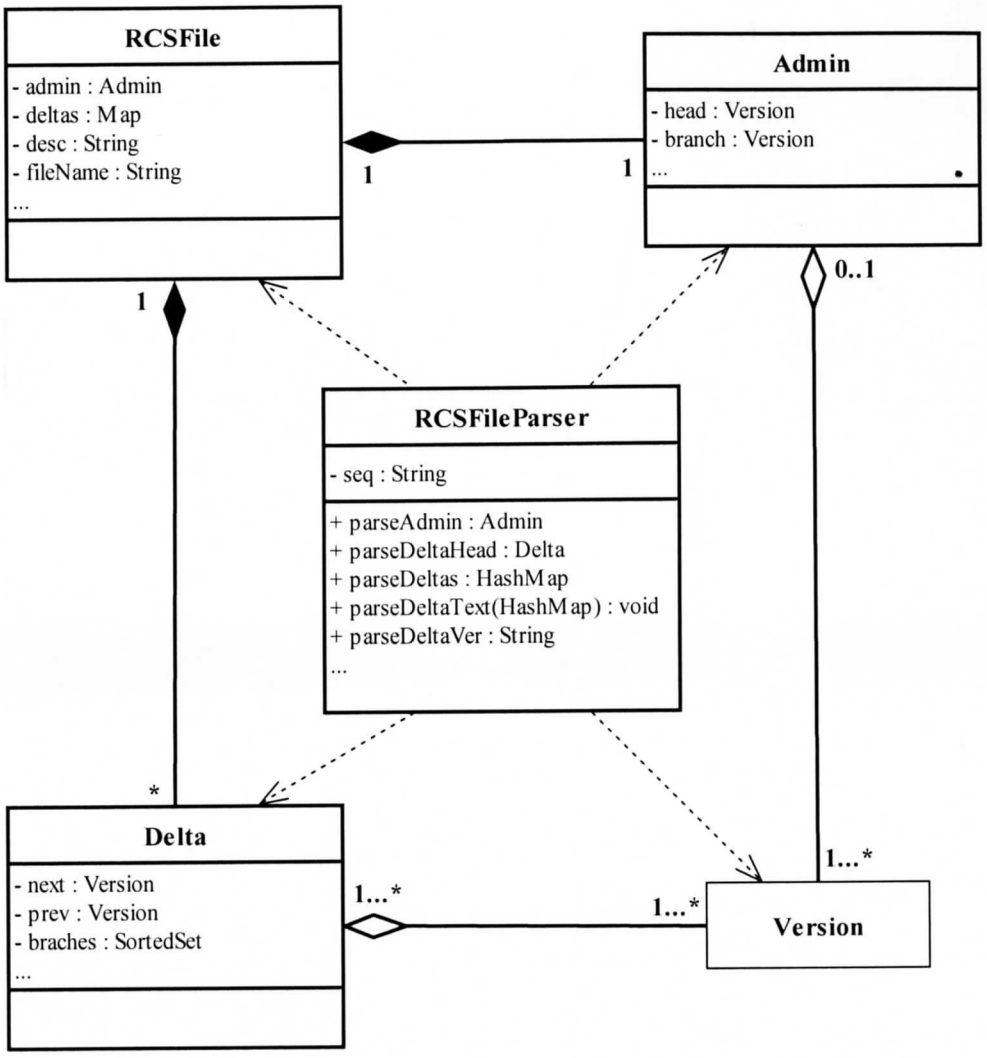

Figure 4.8 Class Diagram of Package vcs

4.2.8. server Package

Package server contains SVCSServerInterface, SVCSServerImpl and SVCSServer three classes.

- SVCSServerInterface is a remote interface that defines the methods the client may call to interact with the server.
- SVCSServerImpl realizes all the methods defined in the SVCSServerInterface.

- SVCSServer initializes a SVCSServerImpl object that is usually called server. SVCSServer defines the main() method to start server application. It accepts the argument of host:port for server binding, where host is the host (remote or local) where the registry is located, port is the port number on which the registry accepts calls. If port is omitted, then the port defaults to 1099 that which is the default port used by RMI registry. If the argument is missing, server will be bound to the *rmiregistry* on the *localhost* with IP 127.0.0.1.

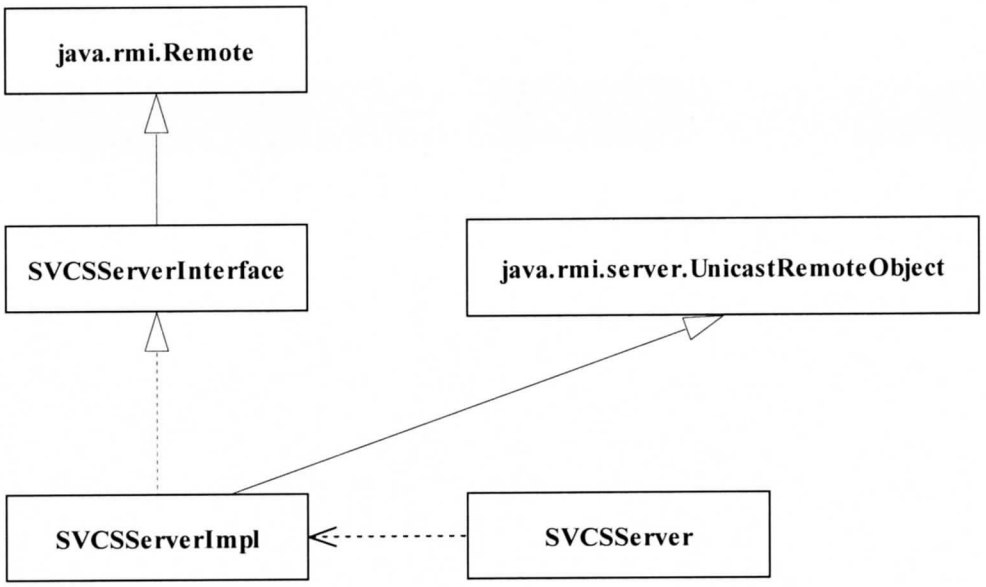

Figure 4.9 Class Diagram of Package server

4.3. Deployment

SVCS is divided into client subsystem and server subsystem, and they should be installed on client machine and server machine respectively. The following steps should be followed in order to run SVCS properly.

- Compile each package by javac command.

- Use rmic to generate stub and/or skeleton of SVCSServerImpl.class.
- SVCSServerImpl_stub.class must be accessible to client. One way to store a copy of SVCSServerImpl_stub.class on the client machine, another way is to use codebase setting to let the client download the stub class dynamically.
- Start rmiregistry
- Start server application by execute SVCSServer. A policy file is required because server has disk read/write operations.
- Start client application by execute SVCSExplorer.

4.4. SVCS Functions and Interfaces

4.4.1. Main Interface

SVCS' main interface is a typical explorer. Its left side is the tree structure of repository, and right side is a table to display information of the revisions under a directory.

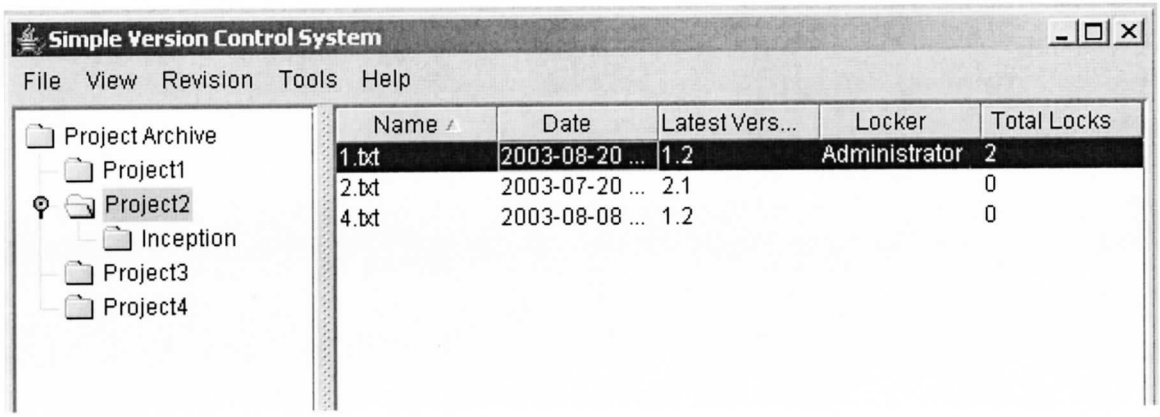

Figure 4.10 SVCS Main Interface

4.4.2. User

SVCS now has no login module. That means any user can connect to the server if the user know the IP address and port of the server. Role-based authentication can be implemented on server end.

The user information is retrieved from system property. SVCS client is recognized as: \\domain\user ID.

4.4.3. Project

4.4.3.1. Add Project

The SVCS files are stored in a tree structure. The directory node is called Project, while the leaf nodes are SVCS files. You can think the project is a directory in repository. When a project starts, it should be added to Project Archive in the left tree pane. Project name and brief comment should be provided. Also created date and creator will be serialized in server for a new project. When client application starts, it will request the information (name, node type, full path) from the persistent storage of a HashTable object stored (RCSProject.fullPath, RcsProject) pairs to display tree graph. On the other hand, if the client adds a new project, the information will be appended to the persistent storage.

Under Project Archive, only project folders can be inserted. SVCS files should be placed under a specific project folder.

Another use of Add Project is to create a sub folder or sub project under an upper level project, which maybe useful to manage the development of a large project.

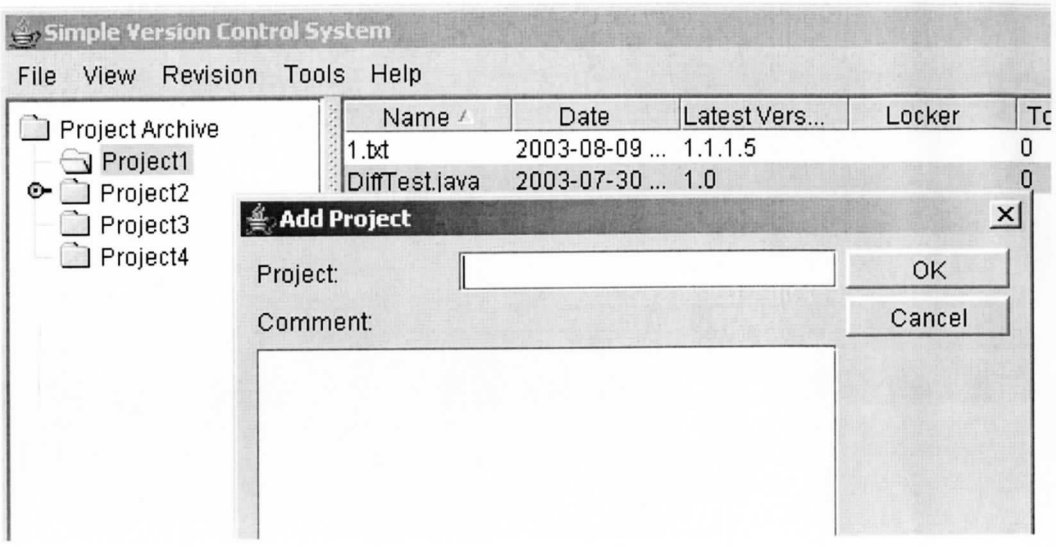

Figure 4.11 Add Project

4.4.3.2. Project Property

After the user select a project, its property can be shown in the dialog like Figure 4.12.

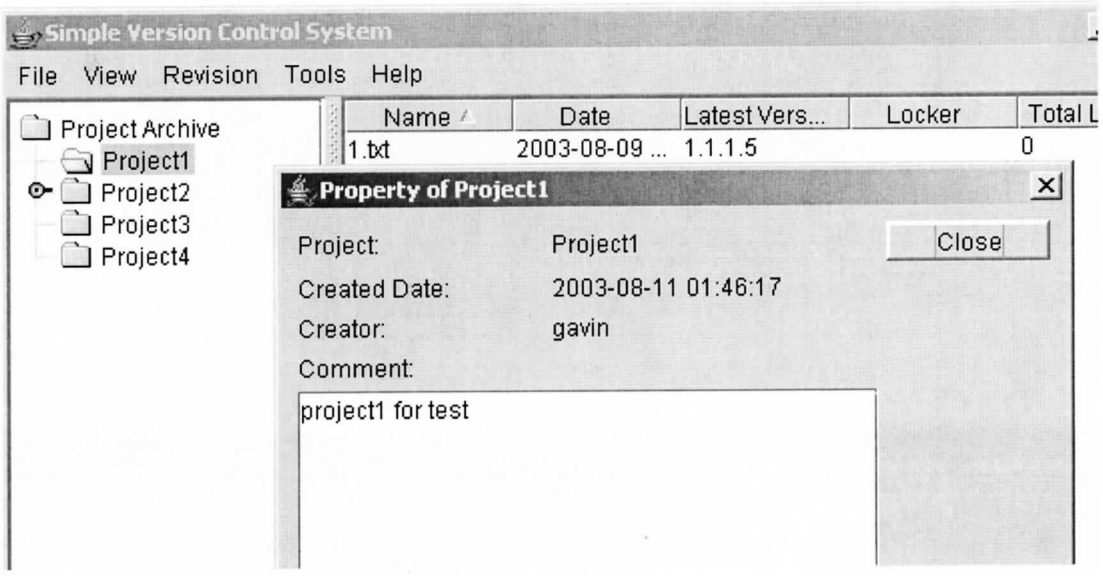

Figure 4.12 Project Property

4.4.3.3. Delete Project

Only the creator of the project has the right to delete his/her own project directory including all the files inside. Other clients cannot delete project.

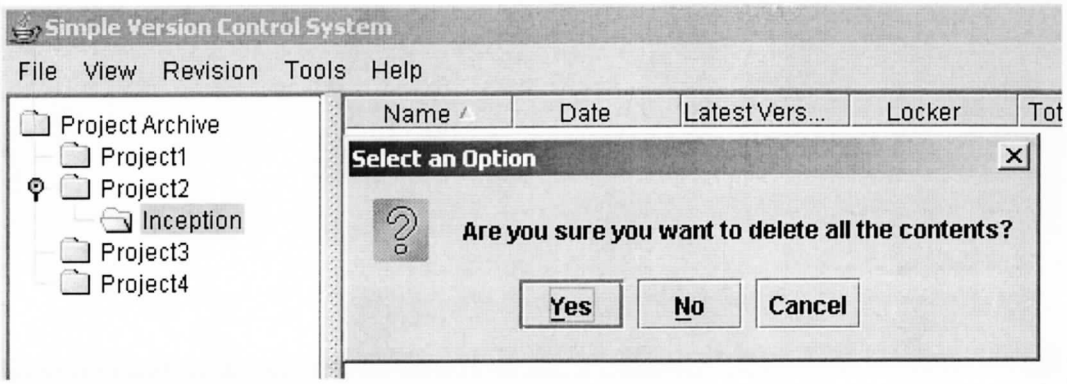

Figure 4.13 Delete Project

4.4.4. Checkout

4.4.4.4. Working Folder

The client can set his/her default working folder for checkout.

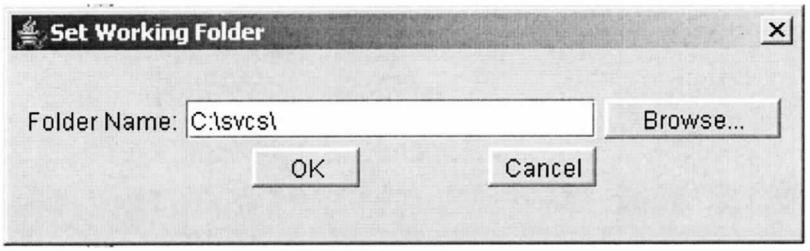

Figure 4.14 Set Working Folder

4.4.4.5. Checkout / View Latest Version

The version retrieved from repository can be read-only or writeable. Destination of the version copy can be set in the dialog shown in Figure 4-15. To retrieve a read-only copy

of a version is called *View,* while *checkout* is to retrieve a writeable copy. If the client viewed a version first, then try to checkout the same version, the writeable copy will replace the read-only one without notice. While in the reverse procedure, confirm dialog will pop out as shown in Figure 4-16.

When a writeable version is checked out, user name will be displayed in *Locker* column, and *Total Locks* will be increased by one.

The Latest version should be retrieved more often than other versions. And use this command may be a little convenient.

The latest version can also be viewed by double-click the selected row in the table pane of main interface.

Multiple files can be retrieved in the same time.

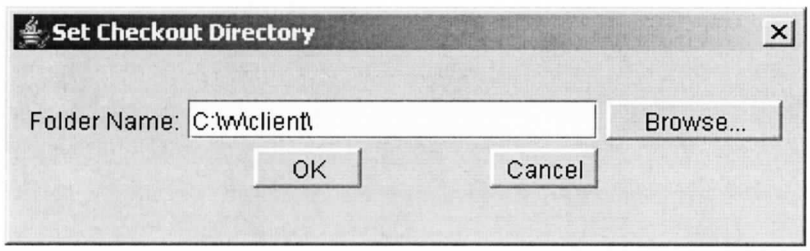

Figure 4.15 Set Checkout Directory

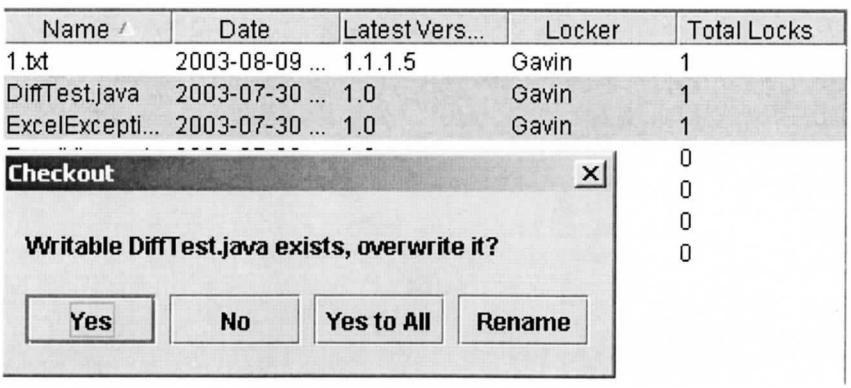

Figure 4.16 Overwrite Confirm Dialog

4.4.4.6. Checkout / View Specific Version

This function retrieves one specific version from repository. All trunk versions and branch versions of a SVCS file will be listed in the dropdown list. Figure 4.17 displays an example.

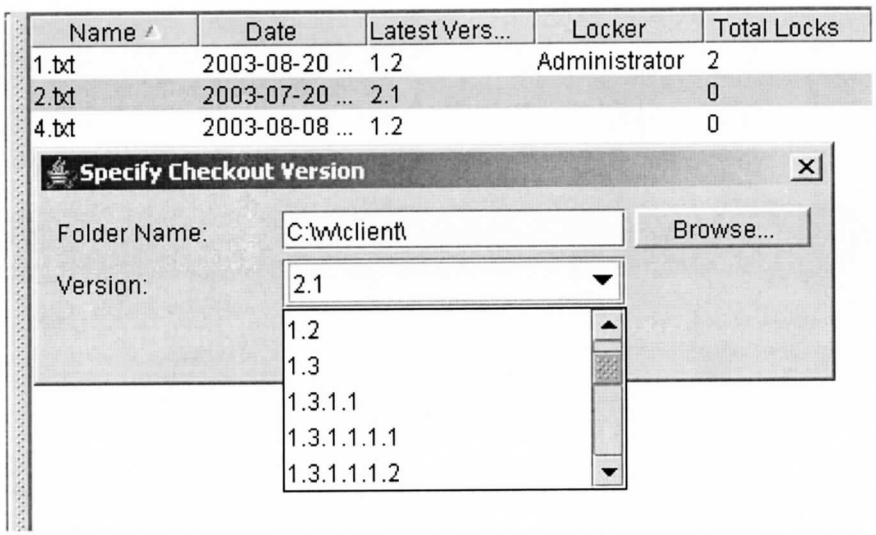

Figure 4.17 Select Specific Version

4.4.5. Check-in

After a version has been checked out and modified, the changed version needed to be checked-in. SVCS will try to pickup the file that locates in the original destination when checked out last time. If the file is found, a dialog will pop out as shown in Figure 4.18.

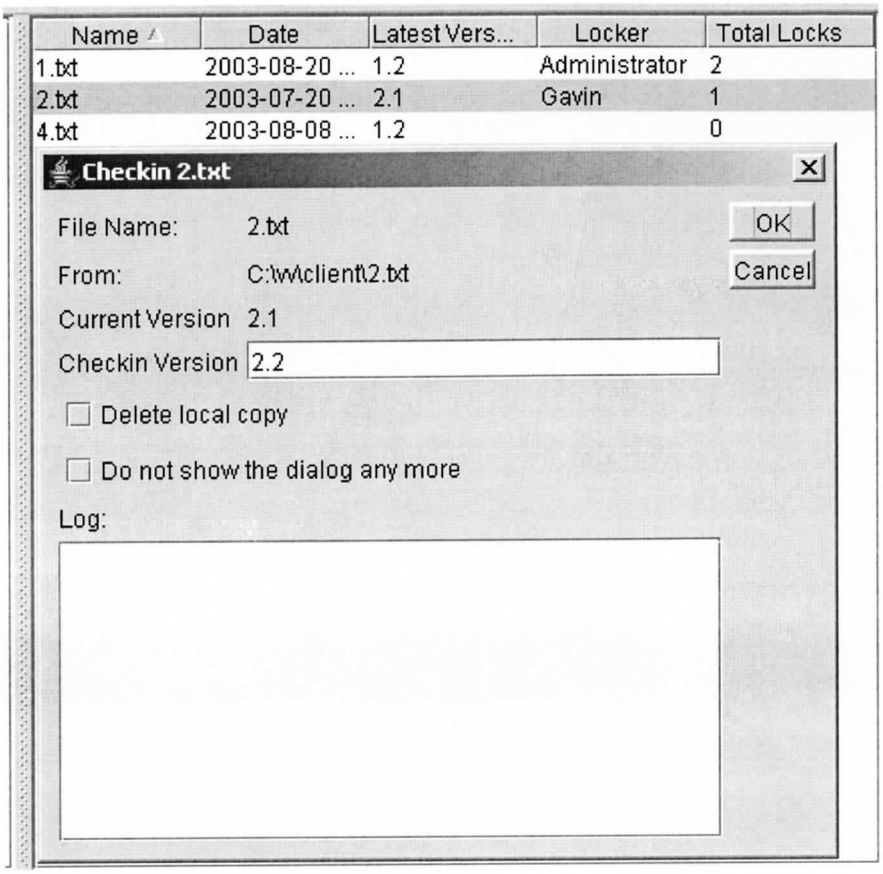

Figure 4.18 Check-in a new Version

If the file cannot be found in the destination where the version has been checked out, a dialog will pop out, as shown in Figure 4.19. And a JFileChooser dialog will be displayed to let the client to choose the file.

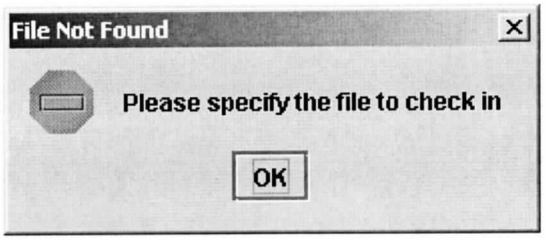

Figure 4.19 File Not Found Dialog

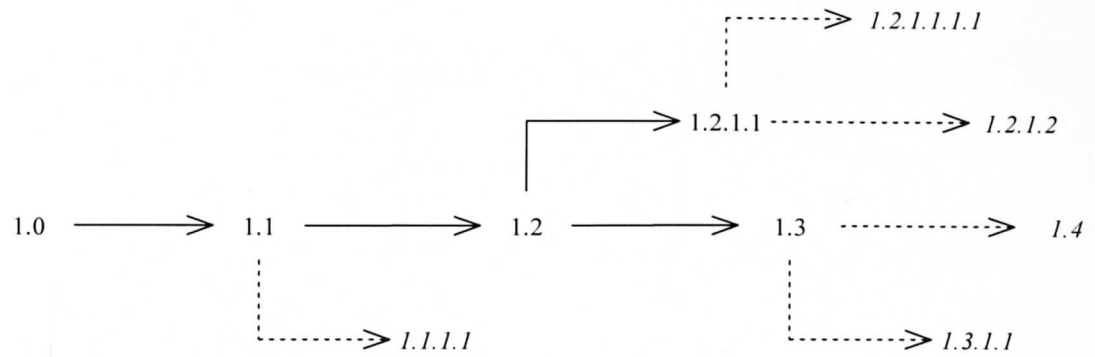

Figure 4.20 Numbering a new Version

A new version number will be given to the version to be checked in. And the client can change the version number to satisfy his real needs. However the version numbering rules must be obeyed, otherwise the check-in command will be canceled. We use the following example to illustrate how to number a version.

In figure 4.20, the versions connected with solid lines are to be checked out. Versions followed by dotted lines are the possible version numbers. There are a branch 1.2.1 and a trunk. If latest trunk version 1.3 is checked out, modified and checked in, the new version can be placed in trunk or in a new branch. Trunk version 1.4 and branch version 1.3.1.1 are two possible choices. If latest branch version 1.2.1.1 is checked out, 1.2.1.2 or 1.2.1.1.1.1 will be the next new version. If previous version 1.1 is checked out, the new version will be 1.1.1.1.

The client can keep the current copy or delete it. And when client check-in multiple files, he can use the same configuration of check-in for each file.

If a user made no modifications and would like to check-in a file, SVCS will keep the original version and abort check-in, as shown in Figure 4.21.

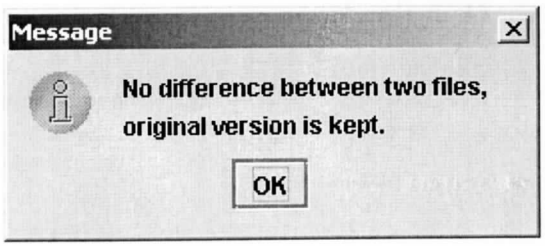

Figure 4.21 No Difference Dialog

After the successful check-in, the values of *Locker* and *Total Locks* column will also be changed.

4.4.6. Branch

4.4.6.7. Set Default Branch

Most time, the client works on the trunk. But sometime a branch needs to be the default working variant. The client can select a version from the list of latest branch versions of all the branches to set default branch. In Figure 4.22, a default branch has been set for the highlighted file.

Name	Date	Latest Vers...	Locker	Total Locks
1.txt	2003-08-09 ...	1.1.2.1		1
DiffTest.java	2003-07-30 ...	1.0	Gavin	1
ExcelExcepti...	2003-07-30 ...	1.0	Gavin	1
ExcelViewer.j...	2003-07-30 ...	1.0		0
ExcelWrappe...	2003-07-30 ...	1.0		0
TestDiff.java	2003-07-30 ...	1.0		0
plan.txt	2003-08-11 ...	1.0		0

Figure 4.22 Set Default Branch

4.4.6.8. Remove Default Branch

The default branch can be removed, and the latest version will switch to the latest trunk version again.

4.4.7. Refresh Archive

When a client made changed to the repository, other clients' main interface won't be updated in time. The client can use this command to update the main interface forcefully.

4.4.8. Sort Table

The files displayed in the table pane can be sorted on each column by selecting the menu or clicking the table header. The order will be changed alternatively between ascending and descending when repeating the command.

4.4.9. Visual Diff

This command is to show the difference between two files visually.

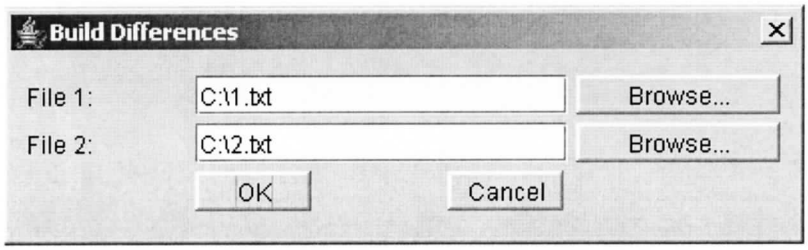

Figure 4.23 Select Two Files

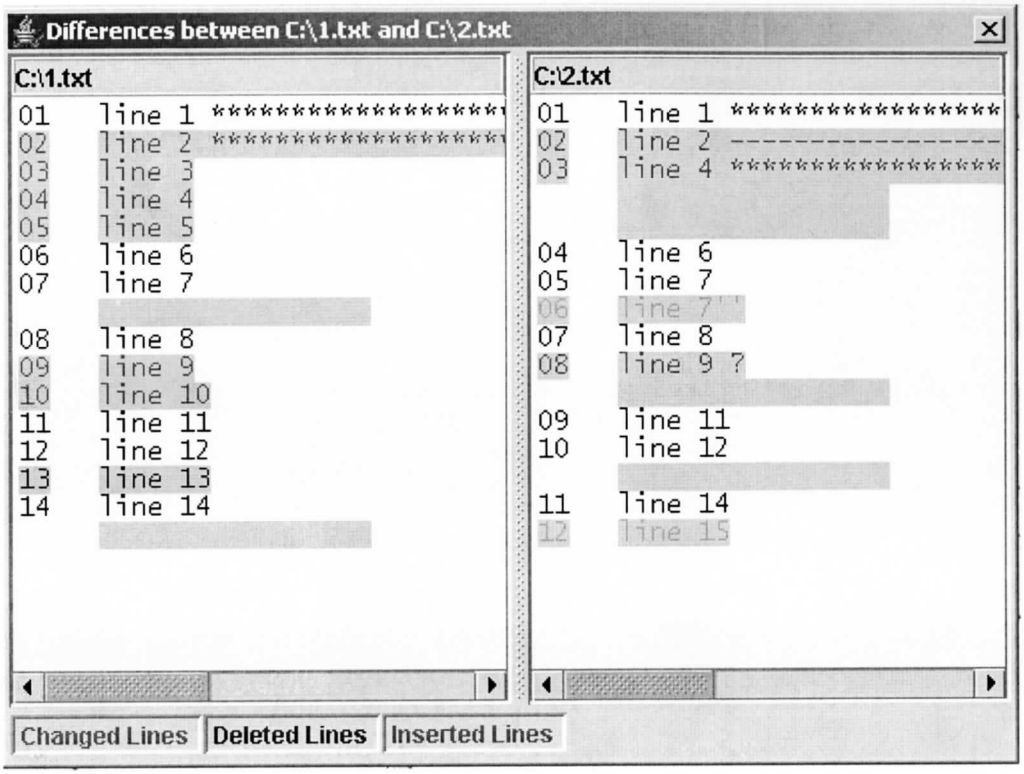

Figure 4.24 Difference Dialog

The left file will be treated as the original file, and right one as the new file. The lines of same contents will be aligned horizontally for two files, and lines in gray background are different. From File 1 to File 2, changed lines are in red color, deleted lines are in blue color, and inserted lines are in green color. The difference dialog and its color scheme are referred to MS Visual Source Safe.

5. Conclusions

SVCS implements most common version control functions based on Java platform. The biggest advantages of SVCS are portability and user friendliness. Its performance is not as good as other applications using native APIs. The architecture of SVCS is successful, and its functionality and performance will be improved in the future.

Reference

[1] Carnegie Mellon/Software Engineering Institute. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1995.

[2] Stephen A. MacKay. The State of the Art in Concurrent, Distributed Configuration Management. In Software Configuration Management: Selected Papers SCM-4 and SCM-5 (Seattle, WA, April), J. Estublier, Ed., LNCS 1005, Springer-Verlag, pp180-194.

[3] Eugene W. Myers. An O(ND) Difference Algorithm and Its Variations. Algorithmica, 1(2): 251-266, 1986.

[4] Marc J. Rochkind. The Source Code Control System. IEEE Transactions on Software Engineering, 1(4): 364-370, Dec 1975.

[5] Walter F. Tichy. Design, Implementation, and Evaluation of a Revision Control System. In Proceedings of the Sixth International Conference on Software Engineering, IEEE computer Society Press, Los Alamitos, CA, pp 56-87.

[6] Walter F. Tichy. RCS-A System for Version Control. Software – Practice and experience, 15(7): 637-654, July 1985.

[7] J. W. Hunt and M. D. McIllroy. An algorithm for Differential file comparison. Tech. Rep. 41, AT&T Bell Laboratories, Inc., Murray Hill, NJ, 1976

[8] J. W. Hunt and T. G. Szymanski. A fast Algorithm for Computing Longest Common Subsequences. Commun. ACM 20(5): 350-353, May, 1977.

[9] Maria Bielikova. Space-efficient Version Storage.

http://www.dcs.elf.stuba.sk/~bielik/scm/delta.html

Appendix A. Quick Guide to Simple Version Control System (SVCS)

1. Requirements

J2SDK 1.4 or above has already been installed on client and server machine.

2. Installation on windows

2.1. Set the value of environment variable *JAVA_HOME* on the client and server machine to be the J2SDK's actual installation directory, for example "C:\j2sdk1.4.2".

2.2. Run *svcs_client.exe* on the **client** computer, it will extract all required client files to the client computer, and generate a directory named "client";

2.3. Run *svcs_server.exe* on the **server** computer, it will extract all required client files to the server computer, and generate a directory named "server";

2.4. Note:

2.4.1. SVCS can be installed with client and server on the same computer.

2.4.2. Suggest to use Sun's JVM, Oracle's rmiregistry doesn't compatible with SVCS.

3. Setup server

3.1. Double click to run *setvcs.bat* under server's installation folder.

3.2. Edit *setServer.bat* under server's installation folder.

3.2.1. If you installed server and client on the different computers, change "127.0.0.1" in the last line to be server's IP address. Otherwise, goto 3.3

3.2.2. Save *setServer.bat*

3.3. Double click to run *setserver.bat*

4. Setup client

4.1. Edit *svcs.bat* under client's installation folder.

4.1.1. If you installed server and client on the different computers, change "127.0.0.1" in the last line to be server's IP address. Otherwise, goto 4.2

4.1.2. Save *svcs.bat*

4.2. Run *svcs.bat*

The Application has a typical explorer GUI interface. Left part is a TreeView Pane, and right part is a TableView(DataGrid) Pane.

5. A Quick Tour

5.1. Add Project

5.1.1. Select "**Project Archive**" Node on the TreeView Pane

5.1.2. Select menu "**File**"-->"**Add Project**".

5.1.3. On the dialog popped up, input "Test" as the project Name, and input some comments. Push "**OK**" button.

5.2. Add Files

5.2.1. Create a source file named as "*Hello.cs*". The contents maybe as follows,

```
public class Hello{
      public static void Main (){
            //output to sonsole
            System.Console.WriteLine("Hello C#!");
      }
}
```

5.2.2. Select "**Test**" Node on the TreeView Pane

5.2.3. Select menu "**File**"-->"**Add Files**"

5.2.4. On the dialog poped up, locate "*Hello.cs*", and push "**Open**" button. "*Hello.cs*" is stored in the Server, and first version is "1.0".

5.2.5. Notice the change on the TableView Pane.

5.2.6. Delete your local "*Hello.cs*" file.

5.2.7. Double click the row of "*Hello.cs*". The contents of selected file will pop up.

5.2.8. Add more files to this project. You can select and add multiple files in the same folder.

5.3. View and Checkout

5.3.1. Here "**View**" means to retrieve a read-only version. "**Checkout**" means to retrieve a writable version.

5.3.2. Select "*Hello.cs*" row on the TableView Pane.

5.3.3. Select menu "**Revision**"-->"**Checkout Latest Version**". A dialog will pop up to set a directory to store the file you are going to retrieve. Use default and Push "**OK**" button.

5.3.4. Go to directory "C:\SVCS", and you'll find "*Hello.cs*".

5.3.5. Edit "*Hello.cs*" and save. The contents now become as follows

```
public class Hello{
      public static void Main (){
            System.Console.WriteLine("Hello C#!");
            System.Console.WriteLine("Hello Innovative Systems!");
            System.Console.WriteLine("Hello everybody!");
      }
}
```

5.4. Checkin

5.4.1. Select "*Hello.cs*" row on the TableView Pane.

5.4.2. Select menu "**Revision**"-->"**Checkin**".

5.4.3. On the dialog popped up, select the checkbox beside "**Delete Local Copy**", and push "**OK**" button.

5.4.4. Notice the changes of each column for the "*Hello.cs*" row.

5.5. Visual Diff

5.5.1. Select "*Hello.cs*" row on the TableView Pane.

5.5.2. Select menu "**Revision**"-->"**View Specific Version**".

5.5.3. On the dialog poped up, select version "1.0" from the dropdown list, and push "**OK**" button.

5.5.4. Rename "*Hello.cs*" to "*Hello10.cs*" under the folder of "C:\SVCS".

5.5.5. Repeat above to retrieve a read-only version "1.1" of "*Hello.cs*", and rename to "*Hello11.cs*".

5.5.6. Select menu "**Tools**"-->"**Visual Diff**".

5.5.7. On the dialog popped up, choose "*Hello10.cs*" and "*Hello11.cs*" as "File 1" and "File 2". Push "**OK**" button.

5.5.8. A Visual Difference between two files will pop up.

5.6. Other Features

5.6.1. Delete Projects. Only the creator of the project can delete it.

5.6.2. Set Working Folder. Set your working space for Checkin and Checkout.

5.6.3. Property. Display properties of Current project. (Will support properties of Current File)

5.6.4. Sort the table view. Click the column header to sort the table.

5.6.5. Refresh Archive. To synchronize the client with the server in a multi-user environment.

5.6.6. Branch Version. Giving a version history of "1.0-->1.1-->1.2-->1.3", if you want to fix a bug in version 1.1, you must create a branch from trunk version 1.1. SVCS can automatically detect and label the branch version. In our example, a 1.1.1 branch will be create, and first version is 1.1.1.1

5.6.7. Set and Remove Branch. "Latest Version" by default is the latest trunk version. But if you are going to develop on a branch, you can set a default branch. And "Latest Version" now will be the latest branch version. Read RSC or CVS manual to get more information about trunk and branch.

5.6.8. Multiple-file Checkin, Checkout or View. Press “Ctrl” or “Shift” key, you can select multiple files/rows on the TableView pane, and you can manipulate (checkin, checkout or view) them simultaneously.

5.6.9. …

6. To-dos

6.1. Toolbar

6.2. Login

6.3. Role-based privilege

6.4. Merge Command

6.5. Improve concurrence and lock

...

Appendix B. Source Code List

\client\AddProjectDialog.java

\client\CheckinDialog.java

\client\DiffDialog.java

\client\MyFileFilter.java

\client\MyMenu.java

\client\MyMenuItem.java

\client\MyTableModel.java

\client\MyTreeRenderer.java

\client\ProjectPropertyDialog.java

\client\Row.java

\client\SetBranchDialog.java

\client\SetCheckoutFolderDialog.java

\client\SetCheckoutVersionDialog.java

\client\SetWorkingFolderDialog.java

\client\ShowFileDialog.java

\client\SVCSExplorer.java

\common\RCSFileInfo.java

\common\RCSProject.java

\common\SVCSServerInterface.java

\common\TableRowObject.java

\common\TreeNodeObject.java

\mydiff\Diff.java

\mydiff\EditScript.java

\mydiff\EditScripts.java

\mydiff\PathPoint.java

\mydiff\PathPoints.java

\server\SVCSServer.java

\server\SVCSServerImpl.java

\server\SVCSServerInterface.java

\showdiff\DiffFrame.java

\showdiff\NoWrapJTextPane.java

\showdiff\ShowDiffDialog.java

\toolkit\Tools.java

\vcs\Admin.java

\vcs\Delta.java

\vcs\RCSFile.java

\vcs\RCSFileParser.java

\vcs\Version.java

Appendix C. Source Code List

```
/*******************************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;

public class AddProjectDialog extends JDialog{
    private Font font;
    private int option=JOptionPane.CLOSED_OPTION;
    private JTextField t1=new JTextField();
    private JTextArea t2=new JTextArea();
    public AddProjectDialog(JFrame f,Font font){
        this(f);
        this.font=font;

    }
    public AddProjectDialog(JFrame f){
        super(f,"Add Project",true);
        Container c=getContentPane();
        c.setLayout(new BorderLayout());

        JPanel p1=new JPanel();
        p1.setLayout(new GridBagLayout());
        p1.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
        c.add(p1,BorderLayout.CENTER);

        GridBagConstraints gc=new GridBagConstraints();
        gc.insets=new Insets(2,2,2,2);
        gc.anchor=GridBagConstraints.WEST;

        JLabel l1=new JLabel("Project:");
        l1.setFont(font);
        p1.add(l1,gc);


        t1.setPreferredSize(new Dimension(150,20));
        gc.gridx=1;
        gc.gridwidth=3;
        gc.weightx=1.0;
        gc.fill=GridBagConstraints.HORIZONTAL;
        p1.add(t1,gc);

        JButton b1=new JButton("OK");
        b1.setFont(font);
        b1.setBorder(new BevelBorder(0));
        gc.gridx=4;
        gc.gridwidth=1;
        gc.fill=GridBagConstraints.HORIZONTAL;
        p1.add(b1,gc);

        JLabel l2=new JLabel("Comment:");
        l2.setFont(font);
        gc.gridx=0;
        gc.gridy=1;
        p1.add(l2,gc);
```

```
        JButton b2=new JButton("Cancel");
        b2.setFont(font);
        b2.setBorder(new BevelBorder(0));
        gc.gridx=4;
        gc.gridwidth=1;
        gc.fill=GridBagConstraints.HORIZONTAL;
        p1.add(b2,gc);


        t2.setLineWrap(true);
        t2.setBorder(new EmptyBorder(new Insets(0,0,0,5)));
        JScrollPane s1=new JScrollPane(t2);
        s1.setPreferredSize(new Dimension(200,60));
        gc.gridx=0;
        gc.gridy=2;
        gc.gridwidth=4;
        gc.gridheight=2;
        gc.fill=GridBagConstraints.BOTH;
        gc.anchor=GridBagConstraints.NORTH;
        gc.weightx=1.0;
        gc.weighty=1.0;
        p1.add(s1,gc);

        b1.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    option=JOptionPane.OK_OPTION;
                    setVisible(false);

                }
            }
        );

        b2.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    option=JOptionPane.CANCEL_OPTION;
                    setVisible(false);
                }
            }
        );
        setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
              (Toolkit.getDefaultToolkit().getScreenSize().height-
320)/2,400,320);
        show();
    }
    public int getOption(){
        return option;
    }
    public String getProjectName(){
        return t1.getText().trim();
    }
    public String getProjectComment(){
        return t2.getText().trim();

    }


}
/*********************************************/
package vcs;

import java.awt.*;
```

```
import java.util.*;
import java.util.regex.*;

import toolkit.*;

//JDK1.4 or higher required to support LinkedHashMap;

public class Admin{
    private Version head;
    private Version branch=null;
    private Vector access=new Vector();
    private Map symbols=Collections.synchronizedMap(new LinkedHashMap());
    private Map locks=Collections.synchronizedMap(new LinkedHashMap());
    private boolean strictLock=true;
    private String comment="# ";
    private String expand=null;
    private Map phrases=Collections.synchronizedMap(new LinkedHashMap());

    public Admin(){
        this(new Version(1,1));
    }
    public Admin(Version ver){
        head=new Version(ver);

    }
    public void setHead(Version ver){
        head=new Version(ver);
    }
    public void setHead(String ver){
        setHead(new Version(ver));
    }
    public Version getHead(){
        return new Version(head);
    }
    public boolean setBranch(Version ver){
        if(ver==null || ver.isBranch()){
            branch=ver;
            return true;
        } else
            return false;


    }
    public boolean setBranch(String ver){
        if(ver==null || ver.length()==0){
            branch=null;
            return true;
        } else
            return setBranch(new Version(ver));
    }
    public Version getBranch(){

        if(branch!=null)
            return new Version(branch);
        else
            return null;

    }
    public boolean addUser(String id){
        if(access.contains(id))
            return false;
        else
            access.add(id);
```

```
        return true;

    }
    public boolean addSymbol(String sym,String ver){
        if(symbols.containsKey(sym))
            return false;
        else
            symbols.put(sym,ver);
        return true;

    }
    public boolean addLock(String id,String ver){
        if(locks.containsKey(ver))
            return false;
        else
            locks.put(ver,id);
        return true;

    }
    public Map getLocks(){
        return locks;

    }
    public boolean removeLock(String ver){
        if(!locks.containsKey(ver))
            return false;
        else
            locks.remove(ver);
        return true;
    }
    public String getLocker(String ver){
        return ((String) locks.get(ver));
    }
    public void setStrictLock(boolean lockType){
        strictLock=lockType;
    }
    public void setComment(String comment){
        this.comment=comment;
    }
    public String getComment(){
        return comment;

    }
    public void setExpand(String expand){
        this.expand=expand;
    }
    public String getExpand(){
        return expand;

    }
    public boolean addPhrase(String value,String key){
        if(phrases.containsKey(key))
            return false;
        else
            phrases.put(key,value);
        return true;

    }

    public String toString(){
        String str="";
        str+="head"+"\t";
```

```
        if(head!=null)
            str+=head.toString();
        str+=";\n";//head

        if(branch!=null)
            str+="branch"+"\t"+branch.toString()+";\n";//branch

        str+="access"; //access
        if(access!=null){
            if(access.size()!=0)

                for(int i=0;i<access.size();i++)
                    str+="\n"+"\t"+((String) access.elementAt(i));
        }
        str+=";\n";

        str+="symbols";
        if(symbols!=null){
            if(symbols.size()!=0){
                Object[] key=symbols.keySet().toArray();
                for(int i=key.length-1;i>=0;i--){

str+="\n"+"\t"+key[i].toString()+":"+symbols.get(key[i]).toString();
                }
            }
        }
        str+=";\n";

        str+="locks";
        if(locks!=null){
            if(locks.size()!=0){
                Object[] key=locks.keySet().toArray();
                for(int i=key.length-1;i>=0;i--){

str+="\n"+"\t"+locks.get(key[i]).toString()+":"+key[i].toString();
                }
            }
        }
        if(strictLock)
            str+="; strict";
        str+=";\n";

        if(comment!=null)
            str+="comment"+"\t"+Tools.RCSString(comment)+";\n";

        if(expand!=null)
            str+="expand"+"\t"+Tools.RCSString(expand)+";\n";

        if(phrases!=null){
            if(phrases.size()!=0){
                str+="phrases";
                Object[] key=phrases.keySet().toArray();
                for(int i=key.length-1;i>=0;i--){
                    str+="\n"+"\t"+key[i].toString()+"
"+phrases.get(key[i]).toString();
                }
            }
        }


        return str;

    }
```

```
}

/*********************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.io.*;

public class CheckinDialog extends JDialog{
    private Font font;
    private int option=JOptionPane.CLOSED_OPTION;
    private JTextField t1=null;
    private JTextArea t2=null;

    private String newVer;
    private String log=null;
    private boolean deleteLocalCopy=false;
    private boolean noCheckinDialog=false;

    private JCheckBox c1,c2;

    public CheckinDialog(JFrame f,Font font,String fileName,String
sourcePath,String orgVer,String newVer){
        this(f,fileName,sourcePath,orgVer,newVer);
        this.font=font;


    }
    public CheckinDialog(JFrame f,String fileName,String sourcePath,String
orgVer,String newVer){
        super(f,"Checkin "+fileName,true);

        this.newVer=newVer;

        Container c=getContentPane();
        c.setLayout(new BorderLayout());

        JPanel p1=new JPanel();
        p1.setLayout(new GridBagLayout());
        p1.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
        c.add(p1,BorderLayout.CENTER);

        GridBagConstraints gc=new GridBagConstraints();
        gc.insets=new Insets(2,2,2,2);
        gc.anchor=GridBagConstraints.WEST;

        JLabel l1=new JLabel("File Name:");
        gc.gridwidth=1;
        l1.setFont(font);
        p1.add(l1,gc);



        JLabel l2=new JLabel(fileName);
        l2.setFont(font);
        gc.gridx=1;
        gc.gridwidth=2;
        gc.fill=GridBagConstraints.HORIZONTAL;
        p1.add(l2,gc);
```

```
JButton b1=new JButton("OK");
b1.setFont(font);
b1.setBorder(new BevelBorder(0));
gc.gridx=6;
gc.gridwidth=1;
p1.add(b1,gc);

JLabel l3=new JLabel("From:");
gc.gridy=1;
gc.gridwidth=1;
gc.gridx=0;
l3.setFont(font);
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l3,gc);

JLabel l4=new JLabel(sourcePath);
l4.setFont(font);
gc.gridx=1;
gc.gridwidth=5;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l4,gc);

JButton b2=new JButton("Cancel");
b2.setFont(font);
b2.setBorder(new BevelBorder(0));
gc.gridx=6;
gc.gridwidth=1;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(b2,gc);


JLabel l5=new JLabel("Current Version");
l5.setFont(font);
gc.gridx=0;
gc.gridy=2;
gc.gridwidth=1;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l5,gc);

JLabel l6=new JLabel(orgVer);
l6.setFont(font);
gc.gridx=1;
gc.gridwidth=5;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l6,gc);

JLabel l7=new JLabel("Checkin Version");
l7.setFont(font);
gc.gridy=3;
gc.gridx=0;
gc.gridwidth=1;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l7,gc);

t1=new JTextField(newVer);
t1.setFont(font);
//t1.setText(newVer);
gc.gridx=1;
gc.gridwidth=5;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(t1,gc);

c1=new JCheckBox("Delete local copy");
```

```
        c1.setFont(font);
        gc.gridx=0;
        gc.gridy=4;
        gc.gridwidth=3;
        //gc.weightx=1.0;
        //ssgc.fill=GridBagConstraints.HORIZONTAL;
        p1.add(c1,gc);

        c2=new JCheckBox("Do not show the dialog any more");
        c2.setFont(font);
        gc.gridx=0;
        gc.gridy=5;
        gc.gridwidth=3;
        //gc.weightx=1.0;
        gc.anchor=GridBagConstraints.WEST;
        p1.add(c2,gc);


        JLabel l11=new JLabel("Log:");
        l11.setFont(font);
        gc.gridy=6;
        gc.gridx=0;
        gc.gridwidth=1;
        gc.fill=GridBagConstraints.HORIZONTAL;
        p1.add(l11,gc);

        t2=new JTextArea();
        t2.setLineWrap(true);
        t2.setBorder(new EmptyBorder(new Insets(0,0,0,5)));
        JScrollPane s1=new JScrollPane(t2);
        s1.setPreferredSize(new Dimension(200,60));
        gc.gridy=7;
        gc.gridx=0;
        gc.gridwidth=6;
        gc.gridheight=5;
        gc.weighty=1.0;
        gc.weightx=1.0;
        gc.fill=GridBagConstraints.BOTH;
        p1.add(s1,gc);

        b1.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                            option=JOptionPane.OK_OPTION;
                        setNewVer();
                        setLog();
                        setVisible(false);


                }
            }
        );

        b2.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    option=JOptionPane.CANCEL_OPTION;
                    setVisible(false);
                }
            }
        );
        setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
(Toolkit.getDefaultToolkit().getScreenSize().height-360)/2,400,360);
```

```
        show();
    }
    public String getNewVer(){
        return newVer;
    }
    public String getLog(){
        if(log!=null && log.length()==0)
            log=null;
        return log;
    }
    public boolean ifDeleteLocal(){
        return c1.isSelected();
    }
    public boolean ifNoCheckinDialog(){
        return c2.isSelected();
    }
    public void setDeleteLocal(boolean b){
        deleteLocalCopy=b;
        c1.setSelected(b);
    }
    private void setLog(){
        log=t2.getText().trim();
    }
    private void setNewVer(){
        newVer=t1.getText().trim();

    }
    public int getOption(){
        return option;
    }



}
/*********************************************/
package vcs;

import java.awt.*;
import java.text.*;
import java.util.*;

import toolkit.*;

public class Delta{
    private Date date=new Date();
    private String author;
    private String state="Exp";
    private String log="*** empty log message ***\n";
    private SortedSet branches=Collections.synchronizedSortedSet(new TreeSet());
    private Version prev=null;
    private Version next=null;
    private Object[] deltaText;
    private DateFormat df=new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss");

    public Delta(){


    }
    public Delta(Date date,String author,String state,String log,Object[]
deltaText){
        this.date=date;
        this.author=author;
        if(state!=null)
```

```
        this.state=state;
    setLog(log);
    this.deltaText=deltaText;
}
public void setDate(Date date){
    this.date=date;
}
public Date getDate(){
    return date;
}
public void setAuthor(String author){
    this.author=author;
}
public String getAuthor(){
    return author;
}
public void setState(String state){
    this.state=state;
}
public String getState(){
    return state;
}
public void setLog(String log){
    if(log==null || log.length()==0)
        this.log="*** empty log message ***\n";
    else this.log=log+"\n";
}
public String getLog(){
    return log.substring(0,log.length()-1);
}
public void addBranch(Version ver){
    branches.add(ver.toString());
}
public void addBranch(String ver){
    branches.add(ver);
}
public Version[] getAllBranches(){
    if(branches==null)
        return null;
    Object[] obj=branches.toArray();
    Version[] ver=new Version[obj.length];
    for(int i=0;i<obj.length;i++)
        ver[i]=new Version((String) obj[i]);
    return ver;
}
public void setPrev(Version prev){
    this.prev=new Version(prev);

}
public void setPrev(String prev){
    this.prev=new Version(prev);

}
public Version getPrev(){
    if(prev==null)
        return null;
    else
        return new Version(prev);

}
public void setNext(Version next){
    this.next=new Version(next);
```

```
    }
    public void setNext(String next){
        this.next=new Version(next);

    }
    public Version getNext(){
        if(next==null)
            return null;
        else
            return new Version(next);

    }
    public void setDeltaText(Object[] obj){

        this.deltaText=obj;
    }


    public Object[] getDeltaText(){

        return deltaText;
    }
    public boolean containsBranch(Version ver){

        return branches.contains(ver.toString());
    }
    public String toString(){
        String str="";
        str+="date\t"+df.format(date)+";";
        str+="\t"+"author "+author+";";
        str+="\t"+"state "+state+";\n";
        str+="branches";
        if(branches!=null)
            for(Iterator i=branches.iterator();i.hasNext();)
                str+="\n\t"+(String) i.next();
        str+=";\n";
        str+="next"+"\t"+((next==null)?"":next.toString())+";\n";
        return str;
    }

    public String deltaString(){
        String str="";
        str+="log"+"\n"+Tools.RCSString(log)+"\n";
        str+="text"+"\n"+Tools.RCSString(deltaText)+"\n";
        return str;

    }

}
/*********************************************/
package mydiff;

import java.awt.*;
import java.util.*;
import java.util.regex.*;

import toolkit.*;

public class Diff{
    private Object[] org,rev;
    private PathPoints vp=null;
    private EditScripts ve=null;
    private final int ADD=0;
```

```
private final int DELETE=1;
private final int CHANGE=2;

public Diff(){};
public Diff(Object[] org,Object[]rev){
    this.org=org;
    this.rev=rev;
    vp=new PathPoints();
    ve=new EditScripts();
}
public Diff(String f1,String f2){
    org=Tools.fileToArray(f1);
    rev=Tools.fileToArray(f2);
}

public PathPoints diff(){
    return diff(org,rev);
}
public PathPoints diff(Object[] org,Object[] rev){
    if(org==null)
        org=new Object[0];
    if(rev==null)
        rev=new Object[0];
    int n=org.length;

    int m=rev.length;
    if((m==0)&&(n==0))
        return vp;
    int max=m+n;
    int size=1+2*max;
    int offset=max;

    int[] v=new int[size];
    v[1+offset]=0;

    int x,y,steps,esType;

    PathPoints ps=new PathPoints();
    PathPoint p;

    stop: for(int d=0;d<max+1;d++){

        for(int k=-d; k<=d; k+=2){

            if((k==-d) || ((k!=d)&&(v[k-1+offset]<v[k+1+offset]))){
                x=v[k+1+offset];
                esType=ADD;
            } else{
                x=v[k-1+offset]+1;
                esType=DELETE;
            }
            y=x-k;
            steps=0;
            while((x<n)&&(y<m)&&(org[x].equals(rev[y]))){
                x++;
                y++;
                steps++;
            }
            v[k+offset]=x;


                p=new PathPoint(new Point(x,y),steps,esType);
                ps.add(p);
```

```
            if((x>=n)&&(y>=m))
                break stop ;
        }
    }

    p=(PathPoint) (ps.lastElement());
    vp.add(p);
    while(ps.size()>0){

        x=p.end.x;
        y=p.end.y;
        for(int i=0;i<p.steps;i++){
            x--; //start of x
            y--; //start of y
        }
        if(p.esType==ADD)
            y--;
        else if(p.esType==DELETE)
            x--;
        ps.removeElementAt(ps.size()-1);
        while(ps.size()>0){
            p=(PathPoint) (ps.lastElement());
            if(p.end.x==x&&p.end.y==y){
                vp.add(p); //add start point

                break;
            }
            ps.removeElementAt(ps.size()-1);
        }



    }

    return vp;

}



public EditScripts getEditScripts(){
    return getEditScripts(vp);
}

public EditScripts getEditScripts(PathPoints vp){
    PathPoint p;
    EditScript es;
    EditScripts tve=new EditScripts();
    int prevEsType=-1;
    int orgFrom,orgTo,revFrom,revTo;
    for(int i=0;i<vp.size()-1;i++){

        p=(PathPoint) (vp.elementAt(i));

        orgTo=p.end.x;
        revTo=p.end.y;


        for(int j=0;j<p.steps;j++){
            orgTo--;
```

```
            revTo--;
        }
        orgFrom=orgTo;
        revFrom=revTo;
        if(p.esType==ADD)
            revFrom--;
        else if(p.esType==DELETE)
            orgFrom--;

        if((p.esType==prevEsType)&&(p.steps==0)){

            es=(EditScript) (tve.lastElement());
            es.orgFrom=orgFrom;
            es.revFrom=revFrom;
            tve.setElementAt(es,tve.size()-1);
        } else if((p.esType!=prevEsType)&&(p.steps==0)&&(prevEsType!=-1)){

            es=(EditScript) (tve.lastElement());
            es.esType=CHANGE;
            es.orgFrom=orgFrom;
            es.revFrom=revFrom;
            tve.setElementAt(es,tve.size()-1);

        } else
            tve.add(new EditScript(orgFrom,revFrom,p.esType,orgTo,revTo));

        prevEsType=p.esType;

    }

    while(tve.size()>0)
        ve.add(tve.remove(tve.size()-1));

    return ve;
}
public String getRCSDiffString(){
    return getRCSDiffString(rev);
}
public String getRCSDiffString(Object[] rev){
    if(ve==null || ve.size()==0)
        return null;
    String str="";

    for(int i=0;i<ve.size()-1;i++)
        str+=((EditScript) (ve.elementAt(i))).toRCSString(rev)+"\n";

    str+=((EditScript) (ve.lastElement())).toRCSString(rev);
    return str;
}
public Object[] getRCSDiffText(){
    return Tools.stringToArray(getRCSDiffString());

}
public static Object[] rcsPatch(Object[] org,Object[] delta){
    String script;
    String es="";
    int from=0;
    int length=0;;
    Vector rev=new Vector();
    int ln=0;
    Pattern p;
    Matcher m;
    if(delta==null)
```

```
            return org;
        for(int i=0;i<delta.length;i++){
            script=(String) delta[i];
            es=script.substring(0,1);
            script=script.substring(1);
            p=Pattern.compile("(\\d+)\\s+(\\d+)");
            m=p.matcher(script);
            if(m.find()){
                from=Integer.parseInt(m.group(1));
                length=Integer.parseInt(m.group(2));

            }
            if(es.equals("d")){
                for(;ln<from-1;ln++)
                    rev.add(org[ln]);
                ln+=length;
            } else if (es.equals("a")){
                for(;ln<from;ln++)
                    rev.add(org[ln]);
                for(int j=0;j<length;j++){
                    i++;
                    rev.add((String) delta[i]);
                }

            }


        }
        return rev.toArray();
    }


}
/*********************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.io.*;

import showdiff.*;

public class DiffDialog extends JDialog{
    private Font font;
    private int option=JOptionPane.CLOSED_OPTION;
    private JTextField t1=new JTextField();
    private JTextField t2=new JTextField();
    private File file1,file2;


    public DiffDialog(JFrame f,Font font){
        this(f);
        this.font=font;


    }
    public DiffDialog(JFrame f){
        super(f,"Build Differences",true);
```

```
Container c=getContentPane();
c.setLayout(new BorderLayout());

JPanel p1=new JPanel();
p1.setLayout(new GridBagLayout());
p1.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
c.add(p1,BorderLayout.CENTER);

GridBagConstraints gc=new GridBagConstraints();
gc.insets=new Insets(2,2,2,2);
gc.anchor=GridBagConstraints.WEST;

JLabel l1=new JLabel("File 1:");
gc.gridwidth=1;
l1.setFont(font);
p1.add(l1,gc);


t1.setMinimumSize(new Dimension(320,20));
gc.gridx=1;
gc.gridwidth=6;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(t1,gc);

JButton b1=new JButton("Browse...");
b1.setFont(font);
b1.setBorder(new BevelBorder(0));
gc.gridx=7;
gc.gridwidth=1;
p1.add(b1,gc);

JLabel l2=new JLabel("File 2:");
l2.setFont(font);
gc.gridx=0;
gc.gridy=1;
gc.gridwidth=1;
l1.setFont(font);
p1.add(l2,gc);


t2.setMinimumSize(new Dimension(320,20));
gc.gridx=1;
gc.gridwidth=6;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(t2,gc);

JButton b2=new JButton("Browse...");
b2.setFont(font);
b2.setBorder(new BevelBorder(0));
gc.gridx=7;
gc.gridwidth=1;
p1.add(b2,gc);


JLabel l3=new JLabel("    ");
gc.gridx=0;
gc.gridy=2;
gc.gridwidth=2;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l3,gc);
```

```
JPanel p2=new JPanel();
gc.gridx=2;
gc.gridwidth=3;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(p2,gc);

JLabel l4=new JLabel("    ");
gc.gridx=5;
gc.gridwidth=3;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l4,gc);


p2.setLayout(new GridLayout(1,3,5,5));
JButton b3=new JButton("OK");
b3.setFont(font);
b3.setBorder(new BevelBorder(0));
p2.add(b3);

p2.add(new JLabel("  "));


JButton b4=new JButton("Cancel");
b4.setFont(font);
b4.setBorder(new BevelBorder(0));
p2.add(b4);



b1.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            selectFile(t1);
        }
    }
);

b2.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            selectFile(t2);
        }
    }
);

b3.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
               showDiff();

        }
    }
);

b4.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            setVisible(false);
        }
    }
);
```

```
        setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
(Toolkit.getDefaultToolkit().getScreenSize().height-120)/2,400,120);
        show();
    }
    public void showDiff(){
        file1=new File(t1.getText().trim());
        file2=new File(t2.getText().trim());

        if(file1.exists() && file2.exists()){
            ShowDiffDialog diff=new ShowDiffDialog(this,file1,file2);
            diff.dispose();
        }
    }
    public void selectFile(JTextField t){
        File f=null;
        JFileChooser chooser=new JFileChooser();
        int answer=chooser.showOpenDialog(this);
        if(answer==JFileChooser.APPROVE_OPTION){
            f=chooser.getSelectedFile();
            t.setText(f.getAbsolutePath());
        }
    }

}
```

Changed Lines

```
/*********************************************/
package mydiff;

import java.awt.*;

public class EditScript extends Object{
    public int orgFrom;
    public int orgTo;
    public int revFrom;
    public int revTo;
    public int esType;
    public final int ADD=0;
    public final int DELETE=1;
    public final int CHANGE=2;

    public EditScript(int orgFrom,int revFrom,int esType,int orgTo,int revTo){
        this.orgFrom=orgFrom;
        this.orgTo=orgTo;
        this.revFrom=revFrom;
        this.revTo=revTo;
        this.esType=esType;
    }

    public String toString(){
        String ec="";
        int orgStart=orgFrom;
        int revStart=revFrom;
        switch(esType){
            case ADD:
                ec="a";
                revStart++;
                break;
            case DELETE:
                ec="d";
                orgStart++;
                break;
            case CHANGE:
```

```
                ec="c";
                revStart++;
                orgStart++;

        }
        String str=Integer.toString(orgStart);
        str+=(orgStart==orgTo)?"":(","+Integer.toString(orgTo));
        str+=ec;
        str+=Integer.toString(revStart);
        str+=(revStart==revTo)?"":(","+Integer.toString(revTo));
        return str;

    }
    public String toRCSString(Object[] rev){
        String str="";

        switch(esType){
            case ADD:

                str="a"+Integer.toString(orgFrom)+" "+Integer.toString(revTo-
revFrom);
                for(int i=revFrom;i<revTo;i++)
                    str+="\n"+((String) rev[i]);
                break;
            case DELETE:

                str="d"+Integer.toString(orgFrom+1)+" "+Integer.toString(orgTo-
orgFrom);

                break;
            case CHANGE:

                str="d"+Integer.toString(orgFrom+1)+" "+Integer.toString(orgTo-
orgFrom)+"\n"+
                    "a"+orgTo+" "+Integer.toString(revTo-revFrom);
                for(int i=revFrom;i<revTo;i++)
                    str+="\n"+((String) rev[i]);

        }
        return str;


    }
}
/*********************************************/
package mydiff;

import java.awt.*;
import java.util.Vector;

public class EditScripts extends Vector{

    public EditScripts(){
        super();
    }

}
/*********************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
import javax.swing.border.*;
import java.io.*;

public class SetCheckoutVersionDialog extends JDialog{
    private Font font;
    private int option=JOptionPane.CLOSED_OPTION;

    private JComboBox c1;
    private JTextField t1=new JTextField();
    private String checkoutFolder;

    private String version;

    public SetCheckoutVersionDialog(JFrame f,Font font,String[] versions,String
workingFolder){
        this(f,versions,workingFolder);
        this.font=font;


    }
    public SetCheckoutVersionDialog(JFrame f,String[] versions,String
workingFolder){
        super(f,"Specify Checkout Version",true);
        this.checkoutFolder=workingFolder;

        Container c=getContentPane();
        c.setLayout(new BorderLayout());

        JPanel p1=new JPanel();
        p1.setLayout(new GridBagLayout());
        p1.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
        c.add(p1,BorderLayout.CENTER);

        GridBagConstraints gc=new GridBagConstraints();
        gc.insets=new Insets(2,2,2,2);
        gc.anchor=GridBagConstraints.WEST;

        JLabel l1=new JLabel("Folder Name:");
        gc.gridwidth=1;
        l1.setFont(font);
        p1.add(l1,gc);


        t1.setMinimumSize(new Dimension(320,20));
        t1.setText(workingFolder);
        gc.gridx=1;
        gc.gridwidth=6;
        gc.weightx=1.0;
        gc.fill=GridBagConstraints.HORIZONTAL;
        p1.add(t1,gc);

        JButton b1=new JButton("Browse...");
        b1.setFont(font);
        b1.setBorder(new BevelBorder(0));
        gc.gridx=7;
        gc.gridwidth=1;
        p1.add(b1,gc);
```

```
JLabel l6=new JLabel("Version:");
l6.setFont(font);
gc.gridx=0;
gc.gridy=1;
gc.gridwidth=1;
p1.add(l6,gc);

c1=new JComboBox(versions);
c1.setFont(font);
c1.setBackground(Color.white);
c1.setMaximumRowCount(5);
c1.setSelectedIndex(versions.length-1);
version=versions[versions.length-1];
c1.addItemListener(
    new ItemListener(){
        public void itemStateChanged(ItemEvent e){
            setVersion();
        }
    }
);
gc.gridx=1;
gc.gridwidth=3;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(c1,gc);

JLabel l3=new JLabel("    ");
gc.gridx=0;
gc.gridy=2;
gc.gridwidth=2;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l3,gc);


JPanel p2=new JPanel();
gc.gridx=2;
gc.gridwidth=3;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(p2,gc);

JLabel l4=new JLabel("    ");
gc.gridx=5;
gc.gridwidth=3;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l4,gc);


p2.setLayout(new GridLayout(1,3,5,5));
JButton b2=new JButton("OK");
b2.setFont(font);
b2.setBorder(new BevelBorder(0));
p2.add(b2);

p2.add(new JLabel("  "));


JButton b3=new JButton("Cancel");
b3.setFont(font);
b3.setBorder(new BevelBorder(0));
p2.add(b3);
```

```
        b1.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    setCheckoutFolder();
                }
            }
        );


        b2.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                       if(!createDir()){
                        option=JOptionPane.OK_OPTION;
                        setVisible(false);
                    }

                }
            }
        );

        b3.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    option=JOptionPane.CANCEL_OPTION;
                    setVisible(false);
                }
            }
        );
        setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
(Toolkit.getDefaultToolkit().getScreenSize().height-120)/2,400,120);
        show();
    }
    public boolean createDir(){
        boolean newDir=false;
        try{
            File f=new File(t1.getText().trim());
            if(!f.exists()){
                int answer=JOptionPane.showConfirmDialog(null,
                    "Checkout folder doesn't exist. Create this folder?","Set
Checkout Folder",
                    JOptionPane.YES_NO_OPTION);
                if(answer==JOptionPane.OK_OPTION)
                    f.mkdir();

                newDir=true;
            }
            checkoutFolder=f.getAbsolutePath();
            if(!checkoutFolder.endsWith(Character.toString(File.separatorChar)))
                checkoutFolder+=File.separatorChar;

            t1.setText(checkoutFolder);

        } catch(Exception e) {};
        return newDir;
    }
    public void setCheckoutFolder(){
        JFileChooser chooser=new JFileChooser();
        chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int answer=chooser.showOpenDialog(this);
        if(answer==JFileChooser.APPROVE_OPTION){
            File f=chooser.getSelectedFile();
            checkoutFolder=f.getAbsolutePath()+File.separatorChar;
```

```
            t1.setText(checkoutFolder);
        }
    }
    public void setVersion(){
        version=(String) c1.getSelectedItem();
        c1.transferFocus();
        //invalidate();

        //if(!t1.getCaret().isVisible())
        //    t1.getCaret().setVisible(true);
        //t1.requestFocus();
        //repaint();
    }
    public int getOption(){
        return option;
    }
    public String getCheckoutFolder(){
        return checkoutFolder;
    }
    public String getCheckoutVersion(){
        return version;
    }


}
/*********************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.io.*;

public class SetWorkingFolderDialog extends JDialog{
    private Font font;
    private int option=JOptionPane.CLOSED_OPTION;
    private JTextField t1=new JTextField();

    private String workingFolder;

    public SetWorkingFolderDialog(JFrame f,Font font,String workingFolder){
        this(f,workingFolder);
        this.font=font;


    }
    public SetWorkingFolderDialog(JFrame f,String workingFolder){
        super(f,"Set Working Folder",true);
        this.workingFolder=workingFolder;
        Container c=getContentPane();
        c.setLayout(new BorderLayout());

        JPanel p1=new JPanel();
        p1.setLayout(new GridBagLayout());
        p1.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
        c.add(p1,BorderLayout.CENTER);

        GridBagConstraints gc=new GridBagConstraints();
        gc.insets=new Insets(2,2,2,2);
        gc.anchor=GridBagConstraints.WEST;

        JLabel l1=new JLabel("Folder Name:");
```

```
gc.gridwidth=1;
l1.setFont(font);
p1.add(l1,gc);


t1.setMinimumSize(new Dimension(320,20));
t1.setText(workingFolder);
gc.gridx=1;
gc.gridwidth=6;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(t1,gc);

JButton b1=new JButton("Browse...");
b1.setFont(font);
b1.setBorder(new BevelBorder(0));
gc.gridx=7;
gc.gridwidth=1;
p1.add(b1,gc);


JLabel l3=new JLabel("   ");
gc.gridx=0;
gc.gridy=1;
gc.gridwidth=2;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l3,gc);


JPanel p2=new JPanel();
gc.gridx=2;
gc.gridwidth=3;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(p2,gc);

JLabel l4=new JLabel("   ");
gc.gridx=5;
gc.gridwidth=3;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l4,gc);


p2.setLayout(new GridLayout(1,3,5,5));
JButton b2=new JButton("OK");
b2.setFont(font);
b2.setBorder(new BevelBorder(0));
p2.add(b2);

p2.add(new JLabel("  "));


JButton b3=new JButton("Cancel");
b3.setFont(font);
b3.setBorder(new BevelBorder(0));
p2.add(b3);


b1.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            setWorkingFolder();
        }
```

```
            }
        );


        b2.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                       if(!createDir()){
                        option=JOptionPane.OK_OPTION;
                        setVisible(false);
                    }

                }
            }
        );

        b3.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    option=JOptionPane.CANCEL_OPTION;
                    setVisible(false);
                }
            }
        );
        setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
(Toolkit.getDefaultToolkit().getScreenSize().height-120)/2,400,120);
        show();
    }
    public boolean createDir(){
        boolean newDir=false;
        try{
            File f=new File(t1.getText().trim());
            if(!f.exists()){
                int answer=JOptionPane.showConfirmDialog(null,
                    "Folder doesn't exist. Create this folder?","Set Working
Folder",
                    JOptionPane.YES_NO_OPTION);
                if(answer==JOptionPane.OK_OPTION)
                    f.mkdir();

                newDir=true;
            }
            workingFolder=f.getAbsolutePath();
            if(!workingFolder.endsWith(Character.toString(File.separatorChar)))
                workingFolder+=File.separatorChar;

            t1.setText(workingFolder);

        } catch(Exception e) {};
        return newDir;
    }
    public void setWorkingFolder(){
        JFileChooser chooser=new JFileChooser();
        chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int answer=chooser.showOpenDialog(this);
        if(answer==JFileChooser.APPROVE_OPTION){
            File f=chooser.getSelectedFile();
            workingFolder=f.getAbsolutePath()+File.separatorChar;
            t1.setText(workingFolder);
        }
    }
    public int getOption(){
```

```
        return option;
    }
    public String getWorkingFolder(){
        return workingFolder;
    }


}
```

Changed Lines

```
/*********************************************/
package client;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

import showdiff.NoWrapJTextPane;
import toolkit.*;

public class ShowFileDialog extends JDialog{
    public ShowFileDialog(JFrame f, String filePath, String fo){
        super(f,filePath,false);
        setResizable(true);
        NoWrapJTextPane t1=new NoWrapJTextPane();
        t1.append(fo);
        JScrollPane s1=new JScrollPane(t1);

        Container c=getContentPane();
        c.setLayout(new BorderLayout());
        c.add(s1,BorderLayout.CENTER);


        setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
              (Toolkit.getDefaultToolkit().getScreenSize().height-
320)/2,400,320);
        show();


    }

}
/*********************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.io.*;
import javax.swing.table.*;
import javax.swing.border.*;
import java.rmi.*;
import java.util.*;

import server.SVCSServerInterface;
import common.*;
import toolkit.*;

public class SVCSExplorer extends JFrame {
```

```
    public static ImageIcon folderIcon;
    public static ImageIcon fileIcon;
    public static ImageIcon folderOpenIcon;
    private ImageIcon ascendingIcon,descendingIcon,nullIcon,logoIcon;

    public static String CHECKOUTVERSIONS="checkoutVersions.dat";
    private String selectedFolder=null;
    private String
workingFolder=System.getProperty("user.dir")+File.separatorChar;
    private String rootPath=null;
    private final String user=System.getProperty("user.name");

    private SVCSServerInterface client;
    private Hashtable projects=new Hashtable();
    private Hashtable checkoutVersions=new Hashtable();
    private DefaultTreeModel treeModel;
    private int lastSelectedPane=-1;
    private final int TABLE=1;
    private final int TREE=2;
    private MyTableModel tableModel;
    private int result=-1;
    private boolean deleteLocal=false;

     private JScrollPane treePane;
    private JScrollPane tablePane;
      private JSplitPane splitPane = new JSplitPane();
    private JMenuBar menuBar;
    private MyMenuItem
addProject,deleteProject,addFiles,setWorkingFolder,exit,property;
    private MyMenuItem
diff,viewLatestVersion,checkoutLatestVersion,setBranch,removeBranch;
    private MyMenuItem byName,byDate,byVersion,byLocker,byLocks,refresh;
      private JTable table;
      private JTree tree;
    private JLabel colLabel;
    public Font headerFont;
    private Border headerBorder;

    private boolean debug=true;

      public SVCSExplorer() {
        super("Simple Version Control System");
            try {
            client=(SVCSServerInterface)
Naming.lookup("//127.0.0.1"+"/SVCSServer");
            init();
            }catch(Exception e) {
            if(debug)
                        e.printStackTrace();
            };
    }

    public SVCSExplorer(String ip) {
        super("Simple Version Control System");
            try {
            client=(SVCSServerInterface) Naming.lookup("//"+ip+"/SVCSServer");
            init();
            }catch(Exception e) {
            if(debug)
                        e.printStackTrace();
            };
    }
```

```
    public static void main(String[] args) {
      SVCSExplorer frame=null;
        if(args.length==0)
              frame = new SVCSExplorer();
      else if(args.length==1)
          frame = new SVCSExplorer(args[0]);
  }

    private void init() throws Exception {
      /*
      //retrive setting of last execution
      System.out.println("workingFolder:
"+System.getProperty("user.workingFolder"));
      if(System.getProperty("user.workingFolder")!=null)
          workingFolder=System.getProperty("user.workingFolder");

      */
      Object obj=client.getRCSProjectInfo();
      if(obj!=null){
          projects=(Hashtable) obj;
          System.out.println("read RCS Project Info OK");
      }
      try{
          File f=new File(CHECKOUTVERSIONS);
          if(f.exists()){
              ObjectInputStream ois=new ObjectInputStream(new
FileInputStream(f));
              obj=ois.readObject();
              if(obj!=null)
                  checkoutVersions=(Hashtable) obj;
              ois.close();
          }
      } catch(Exception e){
          System.err.println("error at get Checkout Verisons information");
      };

         Container c=this.getContentPane();
      //this.setDefaultCloseOperation(EXIT_ON_CLOSE);
          c.setLayout(new BorderLayout());
              splitPane.addComponentListener(new ComponentAdapter() {
                    public void componentResized(ComponentEvent e) {

splitPane.setDividerLocation(10+splitPane.getSize().width/4);
                    }
              }
      );

      try {
                File f=new File("client/images");
          String imagePath = "client/images/";
                folderIcon = new ImageIcon(imagePath + "Folder.gif");
                fileIcon = new ImageIcon(imagePath + "File.gif");
                folderOpenIcon=new ImageIcon(imagePath+"FolderOpen.gif");
          ascendingIcon=new ImageIcon(imagePath+"ascending.gif");
          descendingIcon=new ImageIcon(imagePath+"descending.gif");
          nullIcon=new ImageIcon(imagePath+"nullIcon.gif");
          logoIcon=new ImageIcon(imagePath+"svcs.gif");
          }  catch (Exception e) {System.err.println("no icon"); };

      //---------------------set tree

      //setup tree for projects info
```

```
        treeModel=new DefaultTreeModel(getRoot());
        tree = new JTree(treeModel);
        tree.setCellRenderer(new MyTreeRenderer());
        tree.setEditable(false);

        //allows single selection for tree nodes.
        int m=TreeSelectionModel.SINGLE_TREE_SELECTION;
        tree.getSelectionModel().setSelectionMode(m);

        tree.addFocusListener(
            new FocusAdapter(){
                public void focusGained(FocusEvent e){
                    lastSelectedPane=TREE;
                }
            }
        );

        tree.addTreeSelectionListener(
            new TreeSelectionListener() {
                public void valueChanged(TreeSelectionEvent e) {
                        DefaultMutableTreeNode node =
                        (DefaultMutableTreeNode)
tree.getLastSelectedPathComponent();

                    if(node!=null && node.isRoot()){
                        tableModel.setData(null);
                        tableModel.colNo=0;
                        tableModel.fireTableDataChanged();
                        table.clearSelection();
                        table.getTableHeader().repaint(); //repaint the header
                        table.revalidate();
                        table.repaint(); //display row data
                    } else if(node!=null){
                        TreeNodeObject obj=(TreeNodeObject) node.getUserObject();
                        selectedFolder=obj.fullPath;
                        refreshTable();
                    }
                    lastSelectedPane=TREE;
                      }
            }
        );


           // ------------------table ---------------
        //tableModel=new MyTableModel();
        tableModel=new MyTableModel(this);
        table = new JTable(tableModel);

        table.setShowGrid(false);
        table.setIntercellSpacing(new Dimension(0,0));

        table.addFocusListener(
            new FocusAdapter(){
                public void focusGained(FocusEvent e){

                    table.setSelectionBackground(Color.blue);
                    table.setSelectionForeground(Color.white);
                    lastSelectedPane=TABLE;
                }
                public void focusLost(FocusEvent e){


                    table.setSelectionBackground(Color.lightGray);
```

```
                    table.setSelectionForeground(Color.black);
                }
            }
        );

        table.getTableHeader().addMouseListener(
            new MouseAdapter(){
                public void mouseClicked(MouseEvent event){
                    int tableColumn= table.columnAtPoint(event.getPoint());
                    int modelColumn=
table.convertColumnIndexToModel(tableColumn);
                    sortTable(modelColumn);
                    table.grabFocus();
                        }
                 }
        );

        table.addMouseListener(
            new MouseAdapter(){
                public void mouseClicked(MouseEvent event){
                    if(event.getClickCount()==2){
                        displayFile();
                    }
                }
            }
        );



        //customize header rendering to show ascending or descending symbol
        headerFont=table.getTableHeader().getFont();
        for(int i=0;i<table.getColumnCount();i++){

table.getTableHeader().getColumnModel().getColumn(i).setHeaderRenderer(new
MyHeaderRenderer());
            //table.getColumnModel().getColumn(i).setCellRenderer(new
MyTableCellRenderer());
        }


        //add to scrollpane
        tree.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
        treePane = new JScrollPane(tree);

        tablePane = new JScrollPane(table);
        tablePane.getViewport().setBackground(Color.white);

        //add to splitPane
        c.add(splitPane,BorderLayout.CENTER);
            splitPane.add(treePane, JSplitPane.LEFT);
            splitPane.add(tablePane, JSplitPane.RIGHT);

        //add menu
        menuBar = new JMenuBar();
        setMenu();
            setJMenuBar(menuBar);


        this.addWindowListener(
            new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                    //System.setProperty("user.workingFolder",workingFolder);
                    System.exit(0);
```

```
                }
            }
        );

            setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-600)/2,
              (Toolkit.getDefaultToolkit().getScreenSize().height-
480)/2,600,480);
        show();
      }

    private void setMenu(){
        MenuListener1 menuListener1=new MenuListener1();

        //File
            MyMenu fileMenu = new MyMenu("File",headerFont);
            menuBar.add(fileMenu);

            addProject = new MyMenuItem("Add Project",headerFont);
            addProject.addActionListener(menuListener1);
        fileMenu.add(addProject);

        deleteProject = new MyMenuItem("Delete Project",headerFont);
            deleteProject.addActionListener(menuListener1);
        fileMenu.add(deleteProject);

        addFiles = new MyMenuItem("Add Files",headerFont);
            addFiles.addActionListener(menuListener1);
        fileMenu.add(addFiles);

        fileMenu.add(new JSeparator());
        setWorkingFolder = new MyMenuItem("Set Working Folder",headerFont);
setWorkingFolder.addActionListener(menuListener1);
            fileMenu.add(setWorkingFolder);

        fileMenu.add(new JSeparator());

        property = new MyMenuItem("Property",headerFont);
            property.addActionListener(menuListener1);
        fileMenu.add(property);

        fileMenu.add(new JSeparator());

        exit = new MyMenuItem("Exit",headerFont);
            exit.addActionListener(menuListener1);
        fileMenu.add(exit);

        //view
        MyMenu viewMenu = new MyMenu("View",headerFont);
            menuBar.add(viewMenu);

        MyMenu sort=new MyMenu("Sort",headerFont);

        MenuListener menuListener=new MenuListener();
        byName=new MyMenuItem("By Name",headerFont);
        byName.addActionListener(menuListener);

        byDate=new MyMenuItem("By Date",headerFont);
        byDate.addActionListener(menuListener);

        byVersion=new MyMenuItem("By Version",headerFont);
        byVersion.addActionListener(menuListener);
```

```
        byLocker=new MyMenuItem("By Locker",headerFont);
        byLocker.addActionListener(menuListener);

        byLocks=new MyMenuItem("By Locks",headerFont);
        byLocks.addActionListener(menuListener);

        sort.add(byName);
        sort.add(byDate);
        sort.add(byVersion);
        sort.add(byLocker);
        sort.add(byLocks);

        viewMenu.add(sort);

        refresh=new MyMenuItem("Refresh Archive",headerFont);
        refresh.addActionListener(menuListener);
        viewMenu.add(refresh);


        //Revision
        MyMenu revisionMenu = new MyMenu("Revision",headerFont);
            menuBar.add(revisionMenu);

        viewLatestVersion=new MyMenuItem("View Latest Version",headerFont);
        viewLatestVersion.addActionListener(menuListener1);
        revisionMenu.add(viewLatestVersion);

        checkoutLatestVersion=new MyMenuItem("Checkout Latest
Version",headerFont);
        checkoutLatestVersion.addActionListener(menuListener1);
        revisionMenu.add(checkoutLatestVersion);

        revisionMenu.add(new JSeparator());

        MyMenuItem viewVersion=new MyMenuItem("View Specific
Version",headerFont);
        viewVersion.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    checkoutSpecific(false);
                }
            }
        );
        revisionMenu.add(viewVersion);

        MyMenuItem checkoutVersion=new MyMenuItem("Checkout Specific
Version",headerFont);
        checkoutVersion.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    checkoutSpecific(true);
                }
            }
        );
        revisionMenu.add(checkoutVersion);

        revisionMenu.add(new JSeparator());

        MyMenuItem checkin=new MyMenuItem("Check in",headerFont);
        checkin.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    checkin();
```

```
                }
            }
        );
        revisionMenu.add(checkin);

        revisionMenu.add(new JSeparator());

        setBranch=new MyMenuItem("Set Default Branch",headerFont);
        setBranch.addActionListener(menuListener1);
        revisionMenu.add(setBranch);

        removeBranch=new MyMenuItem("Remove Default Branch",headerFont);
        removeBranch.addActionListener(menuListener1);
        revisionMenu.add(removeBranch);

        //Tool
        MyMenu toolMenu = new MyMenu("Tools",headerFont);
            menuBar.add(toolMenu);

        diff=new MyMenuItem("Visual Diff",headerFont);
        diff.addActionListener(menuListener1);
        toolMenu.add(diff);

        //Help
        MyMenu helpMenu = new MyMenu("Help",headerFont);
            menuBar.add(helpMenu);

        MyMenuItem about=new MyMenuItem("About",headerFont);
        about.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    JOptionPane.showMessageDialog(null,
                        "Simple Verison Control System \nCopyright (c) 2003 Zhu
Wang",
                        "About SVCS",JOptionPane.PLAIN_MESSAGE,logoIcon);
                }
            }
        );
        helpMenu.add(about);
    }
    public boolean displayFile(){
        if(table.getSelectedRowCount()!=1)
            return false;
        int row=table.getSelectedRow();
        String fileName=(String) tableModel.getValueAt(row,0);
        String filePath=selectedFolder+File.separatorChar+fileName;
        try{
            String fileContent=Tools.arrayToString(client.checkout(filePath),
                                System.getProperty("line.separator"));
            ShowFileDialog d=new ShowFileDialog(this,fileName,fileContent);
            //d.dispose();
        } catch(Exception e){
            e.printStackTrace();
            return false;
        };

        return true;
    }
    public void addProject(){
        DefaultMutableTreeNode selectedNode =
                (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
        if(lastSelectedPane==TREE && selectedNode!=null){
            AddProjectDialog d=new AddProjectDialog(this);
```

```
            if(selectedFolder!=null && d.getOption()==JOptionPane.OK_OPTION){
                String name=d.getProjectName();
                String comm=d.getProjectComment();

                RCSProject proj=new RCSProject(name);
                proj.desc=comm;
                proj.creator=System.getProperty("user.name");
                proj.fullPath=selectedFolder+File.separatorChar+proj.name;

                try{
                    if(projects.containsKey(proj.fullPath))
                        return;
                    proj.date=client.addProject(proj.fullPath);
                    TreeNodeObject obj=new TreeNodeObject();
                    obj.name=name;
                    obj.isDir=true;
                    obj.fullPath=proj.fullPath;
                    //insert as the child of the selected Node
                    DefaultMutableTreeNode newNode=new
DefaultMutableTreeNode(obj);

treeModel.insertNodeInto(newNode,selectedNode,selectedNode.getChildCount());
                    //desplay the new node
                    TreeNode[] nodes=treeModel.getPathToRoot(newNode);
                    tree.scrollPathToVisible(new TreePath(nodes));
                    projects.put(proj.fullPath,proj);

                    client.writeRCSProjectInfo(projects);
                } catch (Exception e) {e.printStackTrace();};
            }
            if(d!=null)
                d.dispose();
        }

    }

    public void deleteProject(){
        DefaultMutableTreeNode selectedNode =
                (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
        if(lastSelectedPane==TREE && selectedNode!=null){
            TreeNodeObject nodeObj=(TreeNodeObject) selectedNode.getUserObject();
            Object obj=projects.get(nodeObj.fullPath); //everyone stores the
projects path he created
            if(obj!=null){ //if someone else use your computer by his own
account
                RCSProject proj=(RCSProject) obj;
                if(System.getProperty("user.name")!=proj.creator){
                    JOptionPane.showMessageDialog(null,"You have no right to
delete the item!");

                } else {
                    int answer=JOptionPane.showConfirmDialog(null,
                            "Are you sure you want to delete all the contents?");
                    if(answer ==JOptionPane.OK_OPTION &&
selectedNode.getParent()!=null){
                        projects.remove(proj.fullPath);
                        treeModel.removeNodeFromParent(selectedNode);
                        try{
                            client.writeRCSProjectInfo(projects);
                            client.deleteProject(proj.fullPath);
                        } catch (Exception e) {e.printStackTrace();};
                    }
                }
```

```
            }
        }

    }

    public void setWorkingFolder(){
        SetWorkingFolderDialog d=new
SetWorkingFolderDialog(this,headerFont,workingFolder);
        if(d.getOption()==JOptionPane.OK_OPTION)
            workingFolder=d.getWorkingFolder();
        d.dispose();
    }
    public String setCheckoutFolder(){
        String folder=null;
        SetCheckoutFolderDialog d=new
SetCheckoutFolderDialog(this,headerFont,workingFolder);
        if(d.getOption()==JOptionPane.OK_OPTION)
            folder=d.getCheckoutFolder();
        d.dispose();
        return folder;
    }
    public void showDiffDialog(){
        DiffDialog d=new DiffDialog(this,headerFont);
        d.dispose();
    }
    public void getProperty(){
        DefaultMutableTreeNode selectedNode =
                (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
        if(lastSelectedPane==TREE && selectedNode!=null &&
selectedFolder!=null){
            Object obj=projects.get(selectedFolder);
            if(obj!=null){
                RCSProject proj=(RCSProject) obj;
                ProjectPropertyDialog d=new ProjectPropertyDialog(this,proj);
            }
        } else if(lastSelectedPane==TABLE && table.getSelectedRowCount()==1){ }
    }

    public void addFiles(){
        if(selectedFolder==null || selectedFolder==rootPath)
            return;
        JFileChooser chooser=new JFileChooser();
        chooser.setMultiSelectionEnabled(true);

        MyFileFilter f1=new MyFileFilter("txt","Plain text file (.txt)");
            MyFileFilter f2=new MyFileFilter("c","C souce file (.c)");
            MyFileFilter f3=new MyFileFilter("cpp","c++ source file (.cpp)");
        MyFileFilter f4=new MyFileFilter("java","Java source file (.java)");
            MyFileFilter f5=new MyFileFilter("pas","Pascal/Delphi source file
(.pas)");
            MyFileFilter f6=new MyFileFilter("bas","Basic/VB source file
(.bas)");
            MyFileFilter f7=new MyFileFilter("cs","C# source file (.cs)");
            MyFileFilter f8=new MyFileFilter("pl","Perl source file (.pl)");
            MyFileFilter f9=new MyFileFilter("htm","HTML file (.htm)");
             MyFileFilter f10=new MyFileFilter("ASP","ASP source file (.asp)");
        MyFileFilter f11=new MyFileFilter("XML","XML source file (.xml)");

        chooser.addChoosableFileFilter(f2);
        chooser.addChoosableFileFilter(f3);
        chooser.addChoosableFileFilter(f4);
        chooser.addChoosableFileFilter(f5);
```

```
        chooser.addChoosableFileFilter(f6);
        chooser.addChoosableFileFilter(f7);
        chooser.addChoosableFileFilter(f8);
        chooser.addChoosableFileFilter(f9);
        chooser.addChoosableFileFilter(f10);
        chooser.addChoosableFileFilter(f11);
        chooser.addChoosableFileFilter(f1);

        int answer=chooser.showOpenDialog(this);
        if(answer==JFileChooser.APPROVE_OPTION){
            File[] f=chooser.getSelectedFiles();
            RCSFileInfo fInfo=null;
            Object[] fcontent=null;
            for(int i=0;i<f.length;i++){
                fInfo=new RCSFileInfo(f[i].getName());
                fInfo.fullPath=selectedFolder+File.separatorChar+fInfo.name;
                fInfo.creator=System.getProperty("user.name");
                fInfo.content=Tools.fileToArray(f[i]);
                try{
                    client.addFile(fInfo);
                } catch(Exception e) {
                    if(debug)
                        e.printStackTrace();
                };

            }
        }
        refreshTable();

    }

    //sort the table and maintain the selection
    private void sortTable(int col){
        int[] selectedRow=null;
        int[] dataRow=null;
        if(table.getSelectedRowCount()>0){
            selectedRow=table.getSelectedRows(); //row No. in the view
            dataRow=new int[selectedRow.length];
            for(int i=0;i<dataRow.length;i++)
                dataRow[i]=tableModel.rows[selectedRow[i]].index; //row No. in
the model

        }
        tableModel.sortOn(col);
        if(dataRow!=null){
            table.clearSelection();
            //Row target=new Row();
            Row target=new Row(tableModel);
            for(int i=0;i<dataRow.length;i++){
                target.index=dataRow[i];
                int rowNo=Arrays.binarySearch(tableModel.rows,target);
                if(tableModel.rows[rowNo].index!=dataRow[i])


rowNo=furtherSearch(tableModel.rows,target,rowNo,dataRow[i],0);
                table.addRowSelectionInterval(rowNo,rowNo);
                result=-1;
            }
        }
    }
    //binary search does not guarrantee the correct object, if not further
binary search
```

```
    private int furtherSearch(Row[] rows,Row target,int rowNo,int dataRowNo,int
offset ){
        Row[] prevRows=new Row[rowNo];
        Row[] postRows=new Row[rows.length-rowNo-1];
        System.arraycopy(rows,0,prevRows,0,rowNo);
        System.arraycopy(rows,rowNo+1,postRows,0,rows.length-rowNo-1);
        int i1=Arrays.binarySearch(prevRows,target);
        if(i1>=0 ){
            if(tableModel.rows[i1+offset].index!=dataRowNo)
                furtherSearch(prevRows,target,i1,dataRowNo,offset);
            else
                result=i1+offset;
        }

        i1=Arrays.binarySearch(postRows,target);
        if(i1>=0){
            if(tableModel.rows[i1+rowNo+offset+1].index!=dataRowNo)
                furtherSearch(postRows,target,i1,dataRowNo,offset+rowNo+1);
            else
                result=i1+offset+rowNo+1;
        }
        return result;
    }

    private void refreshTable(){
        table.clearSelection();
        tableModel.reLoadTableData();
        table.getTableHeader().repaint(); //repaint the header, let column 1
display Ascending Status
        table.revalidate();
        table.repaint(); //display row data
    }
    private void refreshTableData(){
            tableModel.reLoadTableData();
            table.revalidate();
               table.repaint(); //display row data
           }
    public void refreshArchive(){
        tree.clearSelection();
        treeModel=new DefaultTreeModel(getRoot());
        tree.setModel(treeModel);
        tree.repaint();
        selectedFolder=null;
        rootPath=null;
        lastSelectedPane=-1;
        //tree.setModel() will fire valueChanged() event, however
lastSelectedPathComponent is null.
        //so table data is not reset. Following is to reset table manually.
        tableModel.setData(null);
        tableModel.colNo=0;
        tableModel.fireTableDataChanged();
        table.clearSelection();
        table.getTableHeader().repaint(); //repaint the header
        table.revalidate();
        table.repaint(); //display row data


    }
    public boolean checkoutLatest(boolean withLock){
        //String filePath;
        int[] rows=null;
        //String fileName;
            //byte[] bytes=null;
```

```
            String fileContent;
            BufferedOutputStream bos=null;
            int answer=-1;
            //File file=null;
        if(lastSelectedPane==TREE && table.getSelectedRowCount()==0){
            rows=new int[table.getRowCount()];
            for(int i=0;i<rows.length;i++)
                rows[i]=i;
        } else if(table.getSelectedRowCount()>0)
                rows=table.getSelectedRows();

        if(rows.length>0){
            String checkoutFolder=setCheckoutFolder();
                  if(checkoutFolder==null)
                return false;

                Object[] options={"Yes","No","Yes to All","Rename"};
            for(int i=0;i<rows.length;i++){
                String locker=(String) tableModel.getValueAt(rows[i],3);
                String fileName=(String) tableModel.getValueAt(rows[i],0);
                String newFileName=fileName;
                if(withLock && locker!=null){
                    if(locker.equals(user))
                        JOptionPane.showMessageDialog(null,
                            "You've already checked out a writable copy of
"+fileName);
                    else
                        JOptionPane.showMessageDialog(null,
                            "A writable copy of "+fileName+
                            " has already been checked out by "+locker);
                    continue;
                }

                              String
filePath=selectedFolder+File.separatorChar+fileName;

                           try{
                                  File file=new File(checkoutFolder+fileName);

                    while(file.exists() && file.canWrite() && answer!=2){
                                        answer=JOptionPane.showOptionDialog(null,
                                               "Writable "+fileName+" exists,
overwrite it?",
                                            "Checkout",JOptionPane.DEFAULT_OPTION,

JOptionPane.PLAIN_MESSAGE,null,options,options[0]);
                                       if(answer==0 || answer==1)
                            break;
                        if(answer==3){
                            newFileName=JOptionPane.showInputDialog(null,
                                "Orginal Name: "+fileName+"\nInput new Name:");
                            file=new File(checkoutFolder+newFileName.trim());
                        }
                    }
                    if(answer==1)
                        continue;
                    if(file.exists()&&file.canRead()&&!file.canWrite())
                        file.delete();
                                bos=new BufferedOutputStream(new
FileOutputStream(file));
                                if(withLock){
                        fileContent=Tools.arrayToString(client.checkout(filePath,
```

```
System.getProperty("user.name")),
                                System.getProperty("line.separator"));
                                    refreshTableData();
                                } else

fileContent=Tools.arrayToString(client.checkout(filePath),
                                System.getProperty("line.separator"));

                    if(fileContent!=null){ //FileOutputStream.close() will
create a empty file
                        byte[] bytes=fileContent.getBytes();
                                    bos.write(bytes);
                                    bos.flush();
                    }
                    if(!withLock)
                                file.setReadOnly();

                    if(bos!=null)
                        bos.close();
                    checkoutVersions.put(user+filePath+

tableModel.getValueAt(rows[i],2),checkoutFolder+newFileName);

                   } catch(Exception e) {
                    if(debug)
                        e.printStackTrace();
                    return false;
                }
            }
        }
        //write the serialized checkout files' info to the disk
        try{
            ObjectOutputStream oos=new ObjectOutputStream(new
FileOutputStream(CHECKOUTVERSIONS));
            oos.writeObject(checkoutVersions);
        } catch(Exception e){
            if(debug)
                e.printStackTrace();
            return false;

        };
        return true;
    }
    public boolean checkoutSpecific(boolean withLock){
            String fileContent;
            BufferedOutputStream bos=null;
            int answer=-1;
        String[] versions;
        String checkoutFolder=null;
        String version=null;

        if(table.getSelectedRowCount()==1){
            int row=table.getSelectedRow();

            String fileName=(String) tableModel.getValueAt(row,0);
            String newFileName=fileName;
                String filePath=selectedFolder+File.separatorChar+fileName;

            try{
                versions=client.getAllVersions(filePath);
                SetCheckoutVersionDialog d=new
SetCheckoutVersionDialog(this,headerFont,
```

```
                                    versions,workingFolder);
            if(d.getOption()==JOptionPane.OK_OPTION){
                checkoutFolder=d.getCheckoutFolder();
                version=d.getCheckoutVersion();
            }
            d.dispose();
        } catch(Exception e){
            if(debug)
                e.printStackTrace();
            return false;
        };

        if(withLock){
            try{

                Map locks=(Map) client.getLocks(filePath);

                String locker=(String) locks.get(version);

                if(locker!=null){
                    if(locker.equals(user)){
                        JOptionPane.showMessageDialog(null,
                            "You've already checked out a writable copy of
"+fileName);
                        return false;
                    } else {
                    JOptionPane.showMessageDialog(null,
                        "A writable copy of "+fileName+
                        " has already been checked out by "+locker);
                    return false;
                    }
                }
            } catch(Exception e){
                if(debug)
                    e.printStackTrace();
                return false;
            };
        }
        if(checkoutFolder==null)
            return false;
        try{
                      File file=new File(checkoutFolder+fileName);
            Object[] options={"Yes","No","Rename"};
            while(file.exists() && file.canWrite()){
                    answer=JOptionPane.showOptionDialog(null,
                            "Writable "+fileName+" exists, overwrite it?",
                                 "Checkout",JOptionPane.DEFAULT_OPTION,

JOptionPane.PLAIN_MESSAGE,null,options,options[0]);
                        if(answer==0 || answer==1)
                    break;
                if(answer==2){
                    newFileName=JOptionPane.showInputDialog(null,
                        "Orginal Name: "+fileName+"\nInput new Name:");
                    file=new File(checkoutFolder+newFileName.trim());
                }
            }

                    if(answer==JOptionPane.NO_OPTION)
                                return false;
            //must delete before init OutputStream due to File.setReadOnly()
            if(file.exists()&&file.canRead()&&!file.canWrite())
                file.delete();
```

```
                bos=new BufferedOutputStream(new FileOutputStream(file));
                                if(withLock){

fileContent=Tools.arrayToString(client.checkout(filePath,version,
                                                    System.getProperty("user.name")),
                            System.getProperty("line.separator"));
                                    refreshTableData();
                                } else

fileContent=Tools.arrayToString(client.checkout(filePath,version,null),
                                System.getProperty("line.separator"));
                if(fileContent!=null){ //FileOutputStream.close() will create a
empty file
                    byte[] bytes=fileContent.getBytes();
                                    bos.write(bytes);
                                    bos.flush();
                }
                if(!withLock)
                                    file.setReadOnly();
                if(bos!=null)
                    bos.close();
                checkoutVersions.put(user+filePath+
                        version,checkoutFolder+newFileName);
                   } catch(Exception e) {
                if(debug)
                    e.printStackTrace();
                return false;
            };
        } else
            return false;

        //write the serialized checkout files' info to the disk
        try{
            ObjectOutputStream oos=new ObjectOutputStream(new
FileOutputStream(CHECKOUTVERSIONS));
            oos.writeObject(checkoutVersions);
        } catch(Exception e){
            if(debug)
                e.printStackTrace();
            return false;
        };
        return true;
    }
    public boolean checkin(){
        int[] rows=null;
        boolean noCheckinDialog=false;
        deleteLocal=false;
        int answer=-1;
        if(lastSelectedPane==TREE && table.getSelectedRowCount()==0){
            rows=new int[table.getRowCount()];
            for(int i=0;i<rows.length;i++)
                rows[i]=i;
        } else if(table.getSelectedRowCount()>0)
                rows=table.getSelectedRows();

        if(rows==null || rows.length==0)
            return false;

        for(int i=0;i<rows.length;i++){
            String fileName=(String) tableModel.getValueAt(rows[i],0);
            String filePath=selectedFolder+File.separatorChar+fileName;
            String[] versions=null;
```

```
            try{
                Map locks=(Map) client.getLocks(filePath);
                if(!locks.containsValue(user))
                    continue;
                LinkedList l=new LinkedList();
                for(Iterator j=locks.entrySet().iterator();j.hasNext();){
                    Map.Entry e=(Map.Entry) j.next();
                    String locker=(String) e.getValue();
                    if(locker.equals(user))
                        l.add((String) e.getKey());

                }

                versions=(String[]) l.toArray(new String[0]);
            } catch(Exception e) {
                if(debug)
                    e.printStackTrace();
                return false;
            };

            if(versions.length>1){
                Object[] options={"Yes","No","Yes to All"};
                for(int j=0;j<versions.length;j++){
                    if(answer!=2){
                            answer=JOptionPane.showOptionDialog(null,
                                    "You have checked out mutiple versions of
"+fileName+". "+
                            "Do you want to check in version "+versions[j],
                                        "Checkin Mutilple
Versions",JOptionPane.DEFAULT_OPTION,

JOptionPane.PLAIN_MESSAGE,null,options,options[0]);
                    }
                    if(answer==1)
                        continue;

noCheckinDialog=showCheckinDialog(noCheckinDialog,fileName,filePath,versions[j]);
                }
            } else if(versions.length==1){

noCheckinDialog=showCheckinDialog(noCheckinDialog,fileName,filePath,versions[0]);
            }
            refreshTableData();
        }
        return true;
    }
    private File specifyFile(){
        JOptionPane.showMessageDialog(null,"Please specify the file to check in",
                        "File Not Found",JOptionPane.ERROR_MESSAGE);
        JFileChooser chooser=new JFileChooser();
        int answer=chooser.showOpenDialog(this);
        if(answer==JFileChooser.APPROVE_OPTION)
            return chooser.getSelectedFile();
        else
            return null;
    }
    private boolean showCheckinDialog(boolean noCheckinDialog,
                    String fileName,String filePath,String version){

        String sourcePath=(String) checkoutVersions.get(user+filePath+version);
        //String sourcePath=(String)
checkoutVersions.remove(user+filePath+version);
        File f=null;
```

```
        int answer=-1;
        if(sourcePath!=null)
            f=new File(sourcePath);
        else {
            f=specifyFile();
            if(f==null)
                return noCheckinDialog;
        }
        while(!f.exists()){
            f=specifyFile();
            if(f==null)
                return noCheckinDialog;
        }
        if(f.exists()){
            Object[] org=Tools.fileToArray(f);
            try{
                sourcePath=f.getAbsolutePath();
                String newVer=client.nextVersion(filePath,version);
                String state=null;
                String log=null;
                if(noCheckinDialog==false){
                    CheckinDialog d=new
CheckinDialog(this,fileName,sourcePath,version,newVer);
                    answer=d.getOption();
                    if(answer==JOptionPane.OK_OPTION){
                        log=d.getLog();
                        newVer=d.getNewVer();
                        deleteLocal=d.ifDeleteLocal();
                        noCheckinDialog=d.ifNoCheckinDialog();
                    }
                }
                if(noCheckinDialog==true || answer==JOptionPane.OK_OPTION){
                    int
result=client.checkin(filePath,org,newVer,version,user,state,log);
                    if(result==1)
                        JOptionPane.showMessageDialog(null,
                            "No difference between two files, \noriginal version
is kept.");
                    if(result!=-1 && deleteLocal)
                        f.delete();
                }
            } catch (Exception e){
                if(debug)
                    e.printStackTrace();
                return noCheckinDialog;
            }
        }
        return noCheckinDialog;
    }
    public boolean setBranch(){
        String fileName,filePath;
        if(table.getSelectedRowCount()==1){
            int row=table.getSelectedRow();

            fileName=(String) tableModel.getValueAt(row,0);

                filePath=selectedFolder+File.separatorChar+fileName;
        } else
            return false;
        String[] versions=null;
        String version=null;
        try{
            versions=client.getAllBranchVersions(filePath);
```

```
            if(versions==null)
                return false;
            SetBranchDialog d=new
SetBranchDialog(this,headerFont,fileName,versions);
            if(d.getOption()==JOptionPane.OK_OPTION){
                version=d.getBranchVersion();
            } else
                return false;
            d.dispose();
            client.setDefaultBranch(filePath,version);
            refreshTableData();
        } catch(Exception e){
            if(debug)
                e.printStackTrace();
            return false;
        };
        return true;
    }
    public boolean removeBranch(){
        String fileName,filePath;
        int row;
        if(table.getSelectedRowCount()==1){
            row=table.getSelectedRow();
            fileName=(String) tableModel.getValueAt(row,0);
                filePath=selectedFolder+File.separatorChar+fileName;
        } else
            return false;
        String ver=(String) tableModel.getValueAt(row,2);
        if(ver.split("\\.").length<=2)
            return false;

        try{
            client.setDefaultBranch(filePath,null);
            refreshTableData();
        } catch(Exception e){
            if(debug)
                e.printStackTrace();
            return false;
        };
        return true;
    }
    private DefaultMutableTreeNode getRoot() {

        DefaultMutableTreeNode root=null;
        try{
            TreeNodeObject nodeObj=new TreeNodeObject();
            nodeObj.name="Project Archive";
            nodeObj.isDir=true;
            nodeObj.fullPath=client.getDirectory();
            rootPath=nodeObj.fullPath;
            root=new DefaultMutableTreeNode(nodeObj);

            Object[] obj=client.getProjectsInfo();
            DefaultMutableTreeNode node;

            for(int i=0;i<obj.length;i++){
                nodeObj=(TreeNodeObject) obj[i];
                node=new DefaultMutableTreeNode(nodeObj);
                root.add(node);
                addChildren(node,nodeObj.fullPath);
            }
        } catch(Exception e){
            if(debug)
```

```
                e.printStackTrace();
            };
            return root;
        }


        private  void addChildren(DefaultMutableTreeNode parent,String path){
            try{
                Object[] obj=client.getFilesInfo(path);
                TreeNodeObject nodeObj=new TreeNodeObject();
                DefaultMutableTreeNode child;
                for(int i=0;i<obj.length;i++){
                    nodeObj=(TreeNodeObject) obj[i];
                    child=new DefaultMutableTreeNode(nodeObj);
                    //only add folder to the Tree
                    if(nodeObj.isDir){
                        parent.add(child);
                        addChildren(child,nodeObj.fullPath);
                    }
                }
            } catch(Exception e) {
                if(debug)
                    e.printStackTrace();
            }
        }

        public Object[] getSelectedRCSFilesInfo() throws RemoteException{
            if(selectedFolder!=null)
                return client.getRCSFilesInfo(selectedFolder);
            else
                return null;

        }

        class MyHeaderRenderer extends DefaultTableCellRenderer{
            public Component getTableCellRendererComponent(JTable table, Object
value,boolean isSelected,
                    boolean hasFocus, int row, int column) {

                        if(column==tableModel.colNo){
                    //Before rendering, isAsc will change to the opposite after the
sorting finished.
                    //Value false means current sorting is in ascending order.
                    if(!tableModel.isAsc)
                                setIcon(ascendingIcon);
                    else
                        setIcon(descendingIcon);
                } else
                    setIcon(nullIcon);
                //set default header properties: font,alignment,border
                setFont(headerFont);
                setText(value.toString());

                setHorizontalTextPosition(JLabel.LEFT);
                setBorder(UIManager.getBorder("TableHeader.cellBorder"));
                setHorizontalAlignment(JLabel.CENTER);

                        return this;

                }
        }
        class MenuListener1 implements ActionListener{
            public void actionPerformed(ActionEvent e){
```

```
            if(e.getSource()==addProject)
                addProject();
            else if(e.getSource()==deleteProject)
                deleteProject();
            else if(e.getSource()==addFiles)
                addFiles();
            else if(e.getSource()==setWorkingFolder)
                setWorkingFolder();
            else if(e.getSource()==property)
                getProperty();
            else if(e.getSource()==exit){
                //System.setProperty("user.workingFolder",workingFolder);
                //System.out.println(System.getProperty("user.workingFolder"));
                System.exit(0);
            }
            else if(e.getSource()==diff)
                showDiffDialog();
            else if(e.getSource()==viewLatestVersion)
                checkoutLatest(false);
            else if(e.getSource()==checkoutLatestVersion)
                checkoutLatest(true);
            else if(e.getSource()==setBranch)
                setBranch();
            else if(e.getSource()==removeBranch)
                System.out.println("removeBranch() success? "+removeBranch());
        }
    };

    class MenuListener implements ActionListener{
        public void actionPerformed(ActionEvent e){
            if(e.getSource()==byName)
                sortTable(0);
            else if(e.getSource()==byDate)
                sortTable(1);
            else if(e.getSource()==byVersion)
                sortTable(2);
            else if(e.getSource()==byLocker)
                sortTable(3);
            else if(e.getSource()==byLocks)
                sortTable(4);
            else if(e.getSource()==refresh)

                refreshArchive();
            table.getTableHeader().repaint();
        }
    };

}
/*********************************************/
package server;

import java.io.*;
import java.rmi.*;

public class SVCSServer {

    public static void main(String args[]){
        try{
        System.setSecurityManager(new RMISecurityManager());
        SVCSServerImpl server=new SVCSServerImpl();
        server.setDirectory("/SVCS/sv/server");
        if(args.length!=1)
            Naming.rebind("//127.0.0.1/SVCSServer",server);
```

```
        else
            Naming.rebind("//"+args[0]+"/SVCSServer",server);
        } catch(Exception e){
            e.printStackTrace();
        }
    }

}
/**********************************************/
package server;

import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

//import client.*;
import vcs.*;
import common.*;

public class SVCSServerImpl extends UnicastRemoteObject implements
SVCSServerInterface {
    private String directory;
    private final String RCSPROJECT=File.separatorChar+"project.dat";
    public SVCSServerImpl() throws RemoteException {
        super();
    }
    public void setDirectory(String directory) throws RemoteException {
        this.directory=directory;
    }
    public String getDirectory() throws RemoteException {
        return (new File(directory)).getAbsolutePath();
    }
    //retrieve RCS projects' first level tree structures
    public Object[] getProjectsInfo() throws RemoteException {

        File f=new File(directory);
        File[] files=f.listFiles();
        Vector v=new Vector();
        TreeNodeObject obj;
        for(int i=0;i<files.length;i++){
            if(files[i].isDirectory()){
                obj=new TreeNodeObject();
                obj.name=files[i].getName();
                obj.isDir=files[i].isDirectory();
                obj.fullPath=files[i].getAbsolutePath();
                v.add(obj);
            }
        }
        return v.toArray();

    }
    //retrieve RCS projects' deeper level tree structures
    public Object[] getFilesInfo(String dir) throws RemoteException {

        File f=new File(dir);
        File[] files=f.listFiles();
        Vector v=new Vector();
        TreeNodeObject obj;
        for(int i=0;i<files.length;i++){
            obj=new TreeNodeObject();
            obj.name=files[i].getName();
            obj.isDir=files[i].isDirectory();
```

```
            obj.fullPath=files[i].getAbsolutePath();
            v.add(obj);
        }
        return v.toArray();

    }
    //retrieve RCSProject properties: name,date,creator,fullPath,desc
    public Object getRCSProjectInfo() throws RemoteException {
        Object obj=null;
        try{
            File f=new File(directory+RCSPROJECT);
            if(f.exists()){
                ObjectInputStream ois=new ObjectInputStream(new
FileInputStream(f));
                obj=ois.readObject();
                ois.close();
            }
        } catch(Exception e){
            System.err.println("error at getRCSProjectInfo()");
        };

        return obj;

    }
    //write serialized HashTable containing RCSProject Objects to the disk
    public void writeRCSProjectInfo(Object obj) throws RemoteException {
        try{
            ObjectOutputStream oos=new ObjectOutputStream(new
FileOutputStream(directory+RCSPROJECT));
            oos.writeObject(obj);
            oos.close();
        } catch(Exception e){
            System.err.println("error at writeRCSProjectInfo()");
        };

    }

    //retrieve RCS file' head version information
    public Object[] getRCSFilesInfo(String dir) throws RemoteException {
        File f=new File(dir);
        File[] files=f.listFiles();
        Vector v=new Vector();
        TableRowObject obj;
        RCSFileParser parser;
        RCSFile rcsFile;
        Admin admin;
        Map deltas;
        Delta delta;
        for(int i=0;i<files.length;i++){
            if(files[i].isFile()){
                try{
                    parser=new RCSFileParser(files[i].getAbsolutePath());
                    rcsFile=parser.parseFile();
                    obj=new TableRowObject();
                    obj.name=files[i].getName();
                    obj.version=rcsFile.getLatestVersion().toString();
                    obj.locker=rcsFile.getAdmin().getLocker(obj.version);
                    //obj.latestTrunk=rcsFile.getAdmin().getHead().toString();

obj.locks=Integer.toString(rcsFile.getAdmin().getLocks().size());
                    deltas=rcsFile.getDeltas();
                    delta=(Delta) deltas.get(obj.version);
                    obj.date=delta.getDate().getTime();
```

```
                v.add(obj);
            } catch (Exception e) {}

        };
    }

    return v.toArray();

}
public Date addProject(String dir) throws RemoteException{
    try{
        File f=new File(dir);
        f.mkdirs();
    }catch(Exception e){
        System.err.println("error at addProject()");
    };
    return new Date();

}
public void deleteProject(String dir) throws RemoteException{
    try{
        File f=new File(dir);


        File[] files=f.listFiles();
        for(int i=0;i<files.length;i++){
            if(files[i].isFile()){
                files[i].delete();
            } else {
                deleteProject(files[i].getAbsolutePath());
            }
        };
        f.delete();

    }catch(Exception e){
        System.err.println("error at deleteProject()");
    };
}
public void addFile(RCSFileInfo fInfo) throws RemoteException{
    File f=new File(fInfo.fullPath);
    if(!f.exists()){
        RCSFile rcsFile=new RCSFile(fInfo.name,"1.0");
        Map deltas=rcsFile.getDeltas();
        Delta delta=(Delta) deltas.get("1.0");
        delta.setAuthor(fInfo.creator);
        delta.setDeltaText(fInfo.content);
        rcsFile.writeTo(fInfo.fullPath);
    }
}

public String[] getAllVersions(String dir) throws RemoteException {
    RCSFileParser parser=new RCSFileParser(dir);
    RCSFile rcsFile=parser.parseFile();
    return rcsFile.getAllVersions();

}
public String[] getAllBranchVersions(String dir) throws RemoteException {
    RCSFileParser parser=new RCSFileParser(dir);
    RCSFile rcsFile=parser.parseFile();
    return rcsFile.getAllBranchVersions();

}
```

```
    public Object getLocks(String dir) throws RemoteException {
        RCSFileParser parser=new RCSFileParser(dir);
        RCSFile rcsFile=parser.parseFile();
        Map locks=rcsFile.getAdmin().getLocks();
        return locks;
    }
    public Object[] checkout(String dir) throws RemoteException {
        return checkout(dir,null);
    }
    public Object[] checkout(String dir,String locker) throws RemoteException {
        RCSFileParser parser=new RCSFileParser(dir);
        RCSFile rcsFile=parser.parseFile();
        String version=rcsFile.getLatestVersion().toString();
        if(locker!=null)
            rcsFile.getAdmin().addLock(locker,version);
        rcsFile.writeTo(dir);
        return rcsFile.getRevision(version);
    }
    public Object[] checkout(String dir,String ver,String locker) throws
RemoteException {
         RCSFileParser parser=new RCSFileParser(dir);
        RCSFile rcsFile=parser.parseFile();
        if(locker!=null)
            rcsFile.getAdmin().addLock(locker,ver);
        rcsFile.writeTo(dir);

        return rcsFile.getRevision(ver);

    }
    //return value:
    //-1:    failure
    //0:    All success
    //1:    No difference, original version kept
    public int checkin(String dir,Object[] org,String newVer,String orgVer,
                String author,String state,String log) throws RemoteException{

        RCSFileParser parser=new RCSFileParser(dir);
        RCSFile rcsFile=parser.parseFile();
        Date date=new Date();
        int result=rcsFile.addRevision(org,orgVer,newVer,date,author,state,log);
        rcsFile.writeTo(dir);
        return result;
    }
    public String nextVersion(String dir,String orgVer) throws RemoteException{
        RCSFileParser parser=new RCSFileParser(dir);
        RCSFile rcsFile=parser.parseFile();
        return rcsFile.nextVersion(orgVer);
    }
    public void setDefaultBranch(String dir,String branch) throws
RemoteException{
        RCSFileParser parser=new RCSFileParser(dir);
        RCSFile rcsFile=parser.parseFile();
        boolean result=rcsFile.getAdmin().setBranch(branch);
        if(result)
            rcsFile.writeTo(dir);

    }
}
/**********************************************/
package server;

import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
import java.io.*;
import java.util.Date;

//import client.*;
import vcs.*;
import common.*;
import toolkit.*;

public interface SVCSServerInterface extends Remote {
    public void setDirectory(String directory) throws RemoteException;
    public String getDirectory() throws RemoteException;

    public Object[] getProjectsInfo() throws RemoteException;
    public Object[] getFilesInfo(String dir) throws RemoteException;
    public Object getRCSProjectInfo() throws RemoteException;
    public void writeRCSProjectInfo(Object obj) throws RemoteException;
    public Object[] getRCSFilesInfo(String dir) throws RemoteException;

    public Date addProject(String dir) throws RemoteException;

    public void deleteProject(String dir) throws RemoteException;
    public void addFile(RCSFileInfo fInfo) throws RemoteException;

    public String[] getAllVersions(String dir) throws RemoteException;
    public String[] getAllBranchVersions(String dir) throws RemoteException;
    public Object getLocks(String dir) throws RemoteException;
    public Object[] checkout(String dir) throws RemoteException;
    public Object[] checkout(String dir,String locker) throws RemoteException;
    public Object[] checkout(String dir,String ver,String locker) throws
RemoteException;
    public int checkin(String dir,Object[] org,String newVer,String orgVer,
                String author,String state,String log) throws RemoteException;
    public String nextVersion(String dir,String orgVer) throws RemoteException;
    public void setDefaultBranch(String dir,String branch) throws
RemoteException;


    //public void uploadFile(FilePacket packet) throws RemoteException;
    //public FilePacket downloadFile(String fileName) throws RemoteException;
}
/*************************************************/
package common;

import java.awt.*;
import java.util.*;
import java.io.*;

public class TableRowObject implements Serializable {
    public String name;
    public long date;
    public String version;
    public String locker;
    //public String latestTrunk;
    public String locks;
}
//RCS projects properties
/*************************************************/
package common;

import java.util.*;
import java.io.*;

public class RCSFileInfo extends RCSProject implements Serializable{
```

```
    /* defined in RCSProject
    public String name;
    public Date date;
    public String desc;
    public String creator;
    public String fullPath;
    */

    public String version;
    public String state;
    public String author;
    public String log;
    public Date lastDate; //date of recent revision


    //lockers[i][0]:version
    //lockers[i][1]:locker's name
    public String[][] Lockers;

    //versionHistory[i][0]:version
    //versionHistory[i][1]:revision date. Long Type since 1970.
Date.getTime().toString();
    public String[][] versionHistory;

    public Object[] content;

    public RCSFileInfo(String name){
        super(name);

    }
}
/*********************************************/
package client;

import javax.swing.JMenuItem;
import java.awt.*;

public class MyMenuItem extends JMenuItem{
    public MyMenuItem(String str,Font font){
        super(str);
        setFont(font);
    }
}
/*********************************************/
package client;

import java.awt.*;
import javax.swing.*;
import java.util.*;
import javax.swing.table.*;

import common.TableRowObject;
import toolkit.*;

public class MyTableModel extends DefaultTableModel {

          private TableRowObject[] data;
        private String[] columnNames = {"Name", "Date", "Latest Version",
"Locker","Total Locks"};
        public Row[] rows;
        public int colNo=0;
        public boolean isAsc=true;
        private SVCSExplorer app;
```

```
public MyTableModel(SVCSExplorer app){
    super();
    this.app=app;
    setData();

}

public void sortOn(int col){
    if(colNo!=col){
        isAsc=true;
        colNo=col;
    }
    if(data!=null){
        for(int i=0;i<rows.length;i++)
            rows[i].asc=isAsc;
        Arrays.sort(rows);
        fireTableDataChanged();
    }
    isAsc=(isAsc)?false:true;
}

public String getColumnName(int column) {
        return columnNames[column];
  }

public int getRowCount() {
    if(data!=null)
            return data.length;
    else
        return 0;
  }

 public int getColumnCount() {
        return columnNames.length;
  }

public Object getValueAt(int row,int col){
    return valueAt(rows[row].index,col);
}

public Object valueAt(int row, int col) {

    TableRowObject record=data[row];
    String str=null;
    switch(col){
        case 0:
                str= record.name;
            break;
        case 1:
            str= Tools.d2k.format(new Date(record.date));
            break;
        case 2:
            str= record.version;
            break;
        case 3:
            if(record.locker!=null)
                str=record.locker;
            break;
        case 4:
            str=record.locks;
            break;
        //case 5:
        //    return record.locks;
```

```
            }
            return str;
          }

         public boolean isCellEditable(int row, int column) {
                return false;
          }

        public void setData() {
            Object[] obj=getTableData();
            if(obj!=null){
                data=new TableRowObject[obj.length];
                rows=new Row[obj.length];
                for(int i=0;i<obj.length;i++){
                    data[i]= (TableRowObject) obj[i];
                    rows[i]=new Row(this);
                    rows[i].index=i;
                }
            } else
                data=null;
            //initial sort information, no matter if data are null.
            colNo=0;
            isAsc=true;
            sortOn(0);
        }
        public void setData(TableRowObject[] data){
            this.data=data;
        }
          private Object[] getTableData() {
            Object[] obj=null;
            try{
                obj=app.getSelectedRCSFilesInfo();

            }catch (Exception e) {e.printStackTrace();};
            return obj;

        }
        public void reLoadTableData(){
            setData();

        }

}
/*********************************************/
package client;

import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.JTree;
import java.awt.Component;
import javax.swing.ImageIcon;
import java.io.*;
import java.awt.Color;

import common.TreeNodeObject;

public class MyTreeRenderer extends DefaultTreeCellRenderer {

  public Component getTreeCellRendererComponent(JTree tree, Object value,
    boolean sel, boolean expanded, boolean leaf, int row, boolean hasFocus) {

    super.getTreeCellRendererComponent(tree, value, sel, expanded, leaf, row,
hasFocus);
```

```
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) value;

        TreeNodeObject f=(TreeNodeObject) node.getUserObject();
        if(f.isDir){
            if(sel)
                setIcon(SVCSExplorer.folderOpenIcon);
            else
                setIcon(SVCSExplorer.folderIcon);
        } else
            setIcon(SVCSExplorer.fileIcon);


        if(sel){
            if(hasFocus){
                setForeground(Color.white);
                setBackgroundSelectionColor(Color.blue);
            } else {
                setForeground(Color.black);
                setBackgroundSelectionColor(Color.lightGray);
            }
        } else{
            setForeground(Color.black);
            setBackgroundSelectionColor(Color.white);
        }

        return this;
      }
}
/*********************************************/
package showdiff;

import java.awt.*;
import java.awt.event.*;
import java.text.*;
import javax.swing.*;
import javax.swing.text.*;

import mydiff.*;

public class NoWrapJTextPane extends JTextPane{
    private int caretPos=0;
    private Document doc;
    private SimpleAttributeSet attr,attrAdd,attrDel,attrChange,attrBlank;
    private final int ADD=0;
    private final int DELETE=1;
    private final int CHANGE=2;

    private String tab="    ";
    private String blank="                    ";
    private String offset=" ";
    private DecimalFormat format;

    public NoWrapJTextPane(){
        super();
        setEditable(false);
        configCaret();
        doc=new DefaultStyledDocument();
        this.setDocument(doc);
        attr=new SimpleAttributeSet();
        StyleConstants.setFontFamily(attr,"Lucida Console");
        StyleConstants.setFontSize(attr,14);
        StyleConstants.setLineSpacing(attr,4);
```

```
        attrAdd=(SimpleAttributeSet) attr.clone();
        StyleConstants.setBackground(attrAdd,Color.lightGray);
        StyleConstants.setForeground(attrAdd,Color.green);

        attrDel=(SimpleAttributeSet) attrAdd.clone();
        StyleConstants.setForeground(attrDel,Color.blue);

        attrChange=(SimpleAttributeSet) attrAdd.clone();
        StyleConstants.setForeground(attrChange,Color.red);

        attrBlank=(SimpleAttributeSet) attrAdd.clone();
        repaint();

    }

    public boolean getScrollableTracksViewportWidth(){
        return (getSize().width<getParent().getSize().width);
    }

    public void setSize(Dimension d){
        if(d.width<getParent().getSize().width)
            d.width=getParent().getSize().width;
        super.setSize(d);
    }

    public void configCaret(){
        addFocusListener(
            new FocusAdapter(){
                public void focusGained(FocusEvent e){

                    if(getCaretPosition()<0)
                        setCaretPosition(caretPos);
                    if(!getCaret().isVisible())
                        getCaret().setVisible(true);

                }
                public void focusLost(FocusEvent e){

                    if(getCaretPosition()>=0)
                        caretPos=getCaretPosition();
                }

            }
        );

    }

    public void append(Object[] obj1,EditScripts ve,boolean isOrg){
        EditScript es;
        int ln=0;
        int s=1;String pattern="0";
        String[] obj=new String[obj1.length];
        String ss;
        for(int i=0;i<obj1.length;i++){
            ss=(String) obj1[i];
            obj[i]=ss.replaceAll("\t","    "); //avoid JTextPane's processing
"\t" and cause line wrap

        }

        while(s*10<obj.length){
            pattern+="0";
```

```
            offset+=" ";
            s*=10;
        }
        format=new DecimalFormat(pattern); // "0000" pattern for alignment

        if(isOrg){
            for(int i=0;i<ve.size();i++){
                es=(EditScript) (ve.elementAt(i));
                switch(es.esType){
                    case ADD:

                        while(ln<es.orgFrom){
                            appendln(ln+1,obj[ln],attr);
                            ln++;
                        }

                        for(int j=0;j<es.revTo-es.revFrom;j++)
                            appendBlankln();
                        break;
                    case DELETE:
                        while(ln<es.orgFrom){
                            appendln(ln+1,obj[ln],attr);
                            ln++;
                        }
                        while(ln<es.orgTo){
                            appendln(ln+1,obj[ln],attrDel);
                            ln++;
                        }
                        break;
                    case CHANGE:
                        while(ln<es.orgFrom){
                            appendln(ln+1,obj[ln],attr);
                            ln++;
                        }
                        while(ln<es.orgTo){
                            appendln(ln+1,obj[ln],attrChange);
                            ln++;
                        }
                        for(int j=0;j<(es.revTo-es.revFrom)-(es.orgTo-
es.orgFrom);j++)
                            appendBlankln();
                }

            }

            while(ln<obj.length){
                appendln(ln+1,obj[ln],attr);
                ln++;

            }
            //append(ln,(String) obj[ln-1],attr);
        } else {
            for(int i=0;i<ve.size();i++){
                es=(EditScript) (ve.elementAt(i));
                switch(es.esType){
                    case ADD:
                        while(ln<es.revFrom){
                            appendln(ln+1,obj[ln],attr);
                            ln++;
                        }
                        while(ln<es.revTo){
                            appendln(ln+1,obj[ln],attrAdd);
                            ln++;
```

```
                    }
                    break;
                case DELETE:
                    while(ln<es.revFrom){
                        appendln(ln+1,obj[ln],attr);
                        ln++;
                    }
                    for(int j=0;j<es.orgTo-es.orgFrom;j++)
                        appendBlankln();
                    break;
                case CHANGE:
                    while(ln<es.revFrom){
                        appendln(ln+1, obj[ln],attr);
                        ln++;
                    }
                    while(ln<es.revTo){
                        appendln(ln+1, obj[ln],attrChange);
                        ln++;
                    }
                    for(int j=0;j<(es.orgTo-es.orgFrom)-(es.revTo-
es.revFrom);j++)
                        appendBlankln();

            }

        }
        while(ln<obj.length){
            appendln(ln+1, obj[ln],attr);
            ln++;
        }
        //append(ln,obj[ln-1],attr);
    }

}
public void append(String line){
    if(line==null)
        line=new String();
    line=line.replaceAll("\t","    "); //avoid JTextPane's processing "\t"
and cause line wrap
    try{
        doc.insertString(doc.getLength(),line,attr);
    } catch(Exception e){};

}
public void append(int ln,String line,SimpleAttributeSet att) {
    try{
        doc.insertString(doc.getLength(),format.format(ln),att);
        doc.insertString(doc.getLength(),tab,attr);
        doc.insertString(doc.getLength(),line,att);
    } catch (Exception e){};
}

public void appendln(int ln,String line,SimpleAttributeSet att) {
    try{
        doc.insertString(doc.getLength(),format.format(ln),att);
        doc.insertString(doc.getLength(),tab,attr);
        doc.insertString(doc.getLength(),line+"\n",att);
    } catch (Exception e){};
}

public void appendBlankln(){
    try{
        doc.insertString(doc.getLength(),offset+tab,attr);
```

```
            doc.insertString(doc.getLength(),blank+"\n",attrBlank);
        } catch (Exception e){};
    }

}

/*************************************************/
package mydiff;

import java.awt.*;
//import java.util.Vector;

public class PathPoint extends Object{
    public Point end;
    public int steps;
    public int esType;

    public PathPoint(Point end,int steps,int esType){
        this.end=end;
        this.steps=steps;
        this.esType=esType;
    }

    public String toString(){
        return end.toString()+","+steps+","+esType;

    }

}
/*************************************************/
package mydiff;

import java.awt.*;
import java.util.Vector;

public class PathPoints extends Vector{

    public PathPoints(){
        super();
    }

}
/*************************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;

import common.RCSProject;
import toolkit.*;

public class ProjectPropertyDialog extends JDialog{
    private Font font;
    private int option=JOptionPane.CLOSED_OPTION;
    private JTextArea t2=new JTextArea();
    private RCSProject proj;
    public ProjectPropertyDialog(JFrame f,RCSProject proj,Font font){
        this(f,proj);
        this.font=font;

    }
```

```
public ProjectPropertyDialog(JFrame f,RCSProject proj){
    super(f,"Property of "+proj.name,true);
    this.proj=proj;

    Container c=getContentPane();
    c.setLayout(new BorderLayout());


    JPanel p1=new JPanel();
    p1.setLayout(new GridBagLayout());
    p1.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
    c.add(p1,BorderLayout.CENTER);

    GridBagConstraints gc=new GridBagConstraints();
    gc.insets=new Insets(2,2,2,2);
    gc.anchor=GridBagConstraints.WEST;

    JLabel l1=new JLabel("Project:");
    l1.setFont(font);
    p1.add(l1,gc);


    JLabel l11=new JLabel(proj.name);
    l11.setFont(font);
    gc.gridx=1;
    gc.gridwidth=3;
    gc.weightx=1.0;
    gc.fill=GridBagConstraints.HORIZONTAL;
    p1.add(l11,gc);

    JButton b1=new JButton("Close");
    b1.setFont(font);
    b1.setBorder(new BevelBorder(0));
    gc.gridx=4;
    gc.gridwidth=1;
    gc.fill=GridBagConstraints.HORIZONTAL;
    p1.add(b1,gc);

    JLabel l2=new JLabel("Created Date:");
    l2.setFont(font);
    gc.gridx=0;
    gc.gridy=1;
    p1.add(l2,gc);


    JLabel l21=new JLabel(Tools.d2k.format(proj.date));
    l21.setFont(font);
    gc.gridx=1;
    gc.gridwidth=3;
    gc.weightx=1.0;
    gc.fill=GridBagConstraints.HORIZONTAL;
    p1.add(l21,gc);


    JLabel l3=new JLabel("Creator:");
    l3.setFont(font);
    gc.gridx=0;
    gc.gridy=2;
    p1.add(l3,gc);

    JLabel l31=new JLabel(proj.creator);
    l31.setFont(font);
    gc.gridx=1;
```

```
        gc.gridwidth=3;
        gc.weightx=1.0;
        gc.fill=GridBagConstraints.HORIZONTAL;
        p1.add(l31,gc);

        JLabel l4=new JLabel("Comment:");
        l4.setFont(font);
        gc.gridx=0;
        gc.gridy=3;
        p1.add(l4,gc);


        t2.setLineWrap(true);
        t2.setBorder(new EmptyBorder(new Insets(0,0,0,5)));
        t2.setText(proj.desc);
        JScrollPane s1=new JScrollPane(t2);
        s1.setPreferredSize(new Dimension(200,60));
        gc.gridx=0;
        gc.gridy=4;
        gc.gridwidth=4;
        gc.gridheight=2;
        gc.fill=GridBagConstraints.BOTH;
        gc.anchor=GridBagConstraints.NORTH;
        gc.weightx=1.0;
        gc.weighty=1.0;
        p1.add(s1,gc);

        b1.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    if(ifChanged()){
                        int answer=JOptionPane.showConfirmDialog(null,
                                    "Save the changes to the Comment?");
                        if(answer==JOptionPane.YES_OPTION)
                            changeComment();
                    }
                    setVisible(false);
                    dispose();


                }
            }
        );

        setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
            (Toolkit.getDefaultToolkit().getScreenSize().height-320)/2,400,320);
        show();
    }
    public boolean ifChanged(){
        return !t2.getText().trim().equals(proj.desc);
    }
    public void changeComment(){
        proj.desc=t2.getText().trim();

    }
}
/*********************************************/
package vcs;

import java.awt.*;
import java.io.*;
import java.util.*;
```

```
import toolkit.*;
import mydiff.*;
import showdiff.*;

public class RCSFile {

    private Admin admin;
    private Map deltas=Collections.synchronizedMap(new HashMap());
    private String desc=null;
    private String fileName;


    public RCSFile(){

        admin=new Admin();
        fileName="untitled.txt";

        //latestVersion=new Version(admin.getHead());

    }
    public RCSFile(String fileName,String ver){
        admin=new Admin(new Version(ver));
        this.fileName=fileName;
        Delta delta=new Delta();
        delta.setLog("Initial revision");
        delta.setDeltaText(Tools.fileToArray(fileName));
        addDelta(ver.toString(),delta);
        //latestVersion=new Version(admin.getHead());
    }
    public RCSFile(String fileName,String ver,String desc){
        admin=new Admin(new Version(ver));
        this.fileName=fileName;
        this.desc=desc+"\n";
        Delta delta=new Delta();
        delta.setDeltaText(Tools.fileToArray(fileName));
        addDelta(ver.toString(),delta);
        //latestVersion=new Version(admin.getHead());
    }
    public void setDesc(String str){
        if(str!=null)
            desc=str+"\n";
    }
    public String getDesc(){
        if(desc!=null)
            return desc.substring(0,desc.length()-1);
        else
            return null;

    }
    public void setFileName(String fileName){
        this.fileName=fileName;
    }

    public void addDelta(String newVersion,Delta delta){
        Version ver=new Version(newVersion);
        if(admin.getBranch()==null&&ver.isTrunk()&&ver.isLarger(admin.getHead()))
            admin.setHead(ver);

        deltas.put(newVersion,delta);

    }
    public void setDeltas(HashMap map){
```

```
    this.deltas=Collections.synchronizedMap(map);

}
public void setAdmin(Admin admin){

    this.admin=admin;

}
public Admin getAdmin(){

    return admin;

}
public Delta getDelta(String ver){
    if(!deltas.containsKey(ver))
        return null;
    return (Delta) deltas.get(ver);

}
public Map getDeltas(){

    return deltas;

}
public String[] getAllVersions(){
    Object[] obj=deltas.keySet().toArray();
    String[] versions=new String[obj.length];
    for(int i=0;i<versions.length;i++)
        versions[i]=(String) obj[i];
    Arrays.sort(versions);
    return versions;
}
public String[] getAllBranchVersions(){
    Object[] obj=deltas.keySet().toArray();
    LinkedList l=new LinkedList();
    for(int i=0;i<obj.length;i++){
        Version ver=new Version((String) obj[i]);
        if(ver.isBranch())
            l.add(ver.toString());
    }
    if(l.size()==0)
        return null;
    String[] versions=(String[]) l.toArray(new String[0]);
    Arrays.sort(versions);
    return versions;
}

public String toString(){
    String str="";
    str+=admin.toString();
    str+="\n\n";
    str+=deltasToString(admin.getHead());
    str+="\n"+"desc"+"\n";
    if(desc!=null)
        str+=Tools.RCSString(desc);
    else
        str+="@@";
    str+="\n";
    str+=deltaTextsToString(admin.getHead());
    return str;
}

public String nextVersion(String orgVer){
```

```
    if(!deltas.containsKey(orgVer))
        return null;
    Version orgVersion=new Version(orgVer);
    Version newVersion=null;
    Delta d=null;
    Version[] branches=null;
    if(orgVersion.isTrunk()){
        int result=orgVersion.compare(admin.getHead());
        if(result==0)
            newVersion=orgVersion.nextVersion();
        else {
            d=(Delta) getDelta(orgVer);
            branches=d.getAllBranches();
            int branchPoint=0;
            for(int i=0;i<branches.length;i++)
                if(branchPoint<branches[i].nthVersion(2))
                    branchPoint=branches[i].nthVersion(2);
            branchPoint++;
            newVersion=orgVersion.addBranch();
            newVersion.setNthVersion(2,branchPoint);
        }
    } else if(orgVersion.isBranch()){
        d=(Delta) getDelta(orgVer);
        if(d.getNext()==null)
            newVersion=orgVersion.nextVersion();
        else{
            branches=d.getAllBranches();
            int branchPoint=0;
            for(int i=0;i<branches.length;i++)
                if(branchPoint<branches[i].nthVersion(orgVersion.size()))
                    branchPoint=branches[i].nthVersion(orgVersion.size());
            branchPoint++;
            newVersion=orgVersion.addBranch();
            newVersion.setNthVersion(orgVersion.size(),branchPoint);
        }
    }
    return newVersion.toString();
}


public String deltasToString(Version ver){

    String str="";

    Vector v=new Vector();
    str+=tranversDeltas(v,ver);

    while(v.size()!=0){
        Vector v2=new Vector();
        while(v.size()!=0)
            str+=tranversDeltas(v2,(Version) v.remove(v.size()-1));

        while(v2.size()!=0)
            v.add(v2.remove(v2.size()-1));

    }

    return str;

}

private String tranversDeltas(Vector v,Version ver){
    String str="";
```

```
        str+=ver.toString()+"\n";
        Delta d=(Delta) deltas.get(ver.toString());
        if(d.getAllBranches().length!=0){
            Version[] vers=d.getAllBranches();
            for(int i=vers.length-1;i>=0;i--)
                v.add(vers[i]);
        }

        str+=d.toString()+"\n";
        if(d.getNext()!=null)
            str+=tranversDeltas(v,d.getNext());

        return str;


    }
    public String deltaTextsToString(Version ver){
        String str="";
        str+="\n\n"+ver.toString()+"\n";
        Delta d=(Delta) deltas.get(ver.toString());
        str+=d.deltaString();
        if(d.getAllBranches().length!=0){
            Version[] vers=d.getAllBranches();
            for(int i=vers.length-1;i>=0;i--)
                str+=deltaTextsToString(vers[i]);
        }
        if(d.getNext()!=null)
            str+=deltaTextsToString(d.getNext());

        return str;



    }
    public Object[] getLatestTrunk(){
        Delta d=(Delta) deltas.get(admin.getHead().toString());
        if(d.getDeltaText()==null)
            return null;
        return (Object[]) d.getDeltaText().clone();

    }
    public Version getLatestVersion(){
        if(admin.getBranch()==null)
            return new Version(admin.getHead());
        String key=admin.getBranch().toString();
        Delta d=(Delta) deltas.get(key);
        while(d.getNext()!=null){
            key=d.getNext().toString();
            d=(Delta) deltas.get(key);
        }
        return new Version(key);

    }
    public Object[] getRevision(Version ver){
        return getRevision(ver.toString());


    }

    public Object[] getRevision(String ver){
        if(!deltas.containsKey(ver))
            return null;
```

```
    Delta d=(Delta) deltas.get(ver);
    Object[] org=getLatestTrunk();
    Object[] rev=org;
    Vector v=new Vector();
    while(d.getPrev()!=null){
        v.add(d.getDeltaText());
        d=(Delta) deltas.get(d.getPrev().toString());
    }

    if(v.size()>0){
        for(int i=v.size()-1;i>=0;i--){
            rev=Diff.rcsPatch(org,(Object[]) v.elementAt(i));
            org=rev;
        }
    }
    return rev;
}
//return value:
//-1:    failure
//0:     All success
//1:     No difference, original version kept
public int addRevision(Object[] org,String orgVer,String newVer,Date date,
                    String author,String state, String log){

    if(!deltas.containsKey(orgVer))
        return -1;
    String locker=(String) admin.getLocker(orgVer);
    if(!locker.equals(author))
        return -1;

    Version newVersion=new Version(newVer);
    Version orgVersion=new Version(orgVer);
    int size=orgVersion.size();
    if(!newVersion.isLarger(orgVersion))
        return -1;
    Object[] rev=getRevision(orgVer);
    Diff diff=null;

    Object[] deltaText=null;
    Delta newDelta=null;
    Delta oldDelta=getDelta(orgVer);
    if(newVersion.isTrunk()){
        if(!orgVer.equals(admin.getHead().toString()))
            return -1;
        if(!newVersion.isLarger(orgVersion))
            return -1;
        diff=new Diff(org,rev);
        diff.diff();
        diff.getEditScripts();


        deltaText=diff.getRCSDiffText();

        if(deltaText==null){
            admin.removeLock(orgVer);
            return 1;
        }

        newDelta=new Delta(date,author,state,log,org);

        newDelta.setNext(orgVersion);
        newDelta.setDeltaText(org);
        oldDelta.setPrev(newVersion);
```

```
            oldDelta.setDeltaText(deltaText);
        } else if(newVersion.isBranch()){
            diff=new Diff(rev,org);
            diff.diff();
            diff.getEditScripts();

            deltaText=diff.getRCSDiffText();
            if(deltaText==null){
                admin.removeLock(orgVer);
                return 1;
            }
            newDelta=new Delta(date,author,state,log,deltaText);

            if(orgVersion.compare(newVersion.getBranchPoint())==0){

                Version[] branches=oldDelta.getAllBranches();
                int branchPoint=newVersion.nthVersion(size);

                //check the branchPoint has been used.
                //1.1 (1.1.2.1, 1.1.4.1) --(nextVersion)-->1.1.5.1
                // while 1.1.1, 1.1.3, 1.1.5 all are valid branch point.

                for(int i=0;i<branches.length;i++)
                    if(branchPoint==branches[i].nthVersion(size))
                        return -1;

                oldDelta.addBranch(newVer);

            }else {
                if(orgVersion.size()!=newVersion.size())
                    return -1;
                else {
                    for(int i=0;i<size-1;i++)
                        if(orgVersion.nthVersion(i)!=newVersion.nthVersion(i))
                            return -1;
                    if(newVersion.nthVersion(size-1)>orgVersion.nthVersion(size-
1))
                        oldDelta.setNext(newVersion);
                    else
                        return -1;
                }
            }
            newDelta.setPrev(orgVersion);
        }
        addDelta(newVer,newDelta);
        admin.removeLock(orgVer);
        return 0;

    }

    public void writeTo(){
        writeTo(fileName);
    }
    public void writeTo(String fileName){
        String str=this.toString();
        byte[] bt=str.getBytes();

        try{
            File f=new File(fileName);
            if(f.canRead() && !f.canWrite())
                f.delete();
            FileOutputStream fo=new FileOutputStream(f);
            fo.write(bt);
```

```
            f.setReadOnly();
            fo.close();
        } catch(Exception e){};
    }
}
/*********************************************/
package client;

import javax.swing.filechooser.FileFilter;
import java.awt.*;
import java.io.*;

public class MyFileFilter extends FileFilter{
    private String description = null;
    private String extension = null;
    public MyFileFilter(String extension, String description) {
        this.description = description;
        this.extension = "." + extension.toLowerCase();
    }
    public String getDescription() {
        return description;
    }
    public boolean accept(File f) {
        if (f == null)
            return false;
        if (f.isDirectory())
            return true;
        return f.getName().toLowerCase().endsWith(extension);
    }
}
/*********************************************/
package client;

import javax.swing.JMenu;
import java.awt.*;

public class MyMenu extends JMenu{
    public MyMenu(String str,Font font){
        super(str);
        setFont(font);
    }
}
/*********************************************/
package vcs;

import java.awt.*;
import java.util.*;
import java.util.regex.*;
import java.io.*;

import toolkit.*;

public class RCSFileParser{
    private String seq;
    private String fileName;
    private String head;

    public RCSFileParser(String fileName){
        seq=Tools.arrayToString(Tools.fileToArray(fileName));
        this.fileName=fileName;
    }
    public RCSFileParser(File file){
        this(file.getName());
```

```
    }
    public RCSFile parseFile(){
        RCSFile f=new RCSFile();
        f.setAdmin(parseAdmin());
        f.setDeltas(parseDeltas());
        f.setDesc(parseDesc());
        parseDeltaText(f.getDeltas());
        return f;


    }
    public void parseDeltaText(Map map){

        String str="";
        int i0=seq.indexOf("desc\n@@\n");
        if(i0==-1)
            i0=simpleParseIndex("desc\n@",seq,"\n@\n")+3;
        else
            i0+=8;
        if(i0!=-1){
            str=seq.substring(i0);
            Pattern p;
            Matcher m;
            String key="";
            String log="";
            String text="@";

            String[] strArray=str.split("@\n\n\n");
            Delta d;
            for(int i=strArray.length-1;i>=0;i--){
                p=Pattern.compile("[\\d+\\.]*\\d+");
                m=p.matcher(strArray[i]);
                if(!m.find()){
                    strArray[i-1]+="@\n\n\n"+strArray[i];
                    continue;
                } else
                    key=m.group();

                log=simpleParse("log\n@",strArray[i],"\n@\n");

                if(log==null){
                    strArray[i-1]+="@\n\n\n"+strArray[i];
                    continue;
                }

                //last @ (delimiter) can be doubled, as Tools.nonRCSString can
single it
                text="@"+simpleParse("\n@\ntext\n@",strArray[i],null)+"@";

                text=Tools.nonRCSString(text);

                d=(Delta) map.get(key);
                d.setLog(log);
                d.setDeltaText(Tools.stringToArray(text));

            }


        }
    }
    public String parseDesc(){
        int i0=seq.indexOf("desc\n@@\n");
```

```
    if(i0==-1){
        String str=simpleParse("desc",seq,"\n@\n");
        str+="@";
        return (Tools.nonRCSString(str));
    } else
        return null;


}
public HashMap parseDeltas(){
    HashMap hm=new HashMap();
    String str="";
    int i0=simpleParseIndex("comment",seq,"@;\n\n");
    if(i0==-1)
        i0=simpleParseIndex("locks",seq,";\n");
    int i1=seq.indexOf(head,i0);
    int i2=seq.indexOf("desc\n",i1);

    if(i2!=-1&&i1!=-1){
        str=seq.substring(i1,i2);
        String[] strArray=str.split(";\n\n");
        Delta d;
        String key;
        String s;
        for(int i=0;i<strArray.length-1;i++){
            s=strArray[i].trim();
            key=parseDeltaVer(s);

            d=parseDeltaHead(s);
            hm.put(key,d);
        }


    }

    parsePrev(hm,head);

    return hm;



}
private void parsePrev(HashMap m,String ver){
    Delta d=(Delta) m.get(ver);
    if(d.getNext()!=null){
        ((Delta) m.get(d.getNext().toString())).setPrev(new Version(ver));
        parsePrev(m,d.getNext().toString());
    }
    Version[] vers=d.getAllBranches();
    if(vers!=null && vers.length!=0){
        for(int i=0;i<vers.length;i++){
            ((Delta) m.get(vers[i].toString())).setPrev(new Version(ver));
            parsePrev(m,vers[i].toString());
        }
    }

}

public String parseDeltaVer(String base){
    return base.substring(0,base.indexOf("\ndate")).trim();

}
public Delta parseDeltaHead(String base){
```

```
    Delta delta=new Delta();
    base=base.substring(base.indexOf("\ndate")+5);
    String[] strArray2=base.split(";");

    delta.setDate(Tools.parseDate(strArray2[0].trim()));

    delta.setAuthor(simpleParse("author",strArray2[1],null));

    delta.setState(simpleParse("state",strArray2[2],null));

    Pattern p=Pattern.compile("[\\d+\\.]*\\d+");
    Matcher m=p.matcher(strArray2[3]);
    while(m.find())
        delta.addBranch(m.group());

    m=p.matcher(strArray2[4]);
    while(m.find())
        delta.setNext(m.group());

    return delta;


}
public Admin parseAdmin(){
    Admin admin=new Admin();

    String str,base;
    String[] strArray,strArray2;
    int i1=seq.indexOf("comment");

    base=seq.substring(0,i1);
    admin.setHead(simpleParse("head",base));
    head=admin.getHead().toString();

    str=simpleParse("branch",base);
    if(str!=null)
        admin.setBranch(str);

    str=simpleParse("access",base);
    if(str.trim().length()!=0){
        strArray=str.split("\n");
        for(int i=0;i<strArray.length;i++)
            admin.addUser(strArray[i].trim());
    }

    str=simpleParse("symbols",base);
    if(str.trim().length()!=0){
        strArray=str.split("\n");
        for(int i=0;i<strArray.length;i++){
            strArray2=strArray[strArray.length-i-1].split(":");
            admin.addSymbol(strArray2[0].trim(),strArray2[1].trim());
        }
    }

    str=simpleParse("locks",base);
    if(str.indexOf("; strict")!=-1){
        admin.setStrictLock(true);
        str=str.replaceAll("; strict","");
    }

    if(str.trim().length()!=0){
        strArray=str.split("\n");
```

```
            for(int i=0;i<strArray.length;i++){
                strArray2=strArray[strArray.length-i-1].split(":");
                admin.addLock(strArray2[0].trim(),strArray2[1].trim());
            }
        }


        int i2=seq.indexOf(admin.getHead().toString(),i1);
        if(i2!=-1){
            base=seq.substring(i1,i2).trim()+"\n";
            admin.setComment(Tools.nonRCSString(simpleParse("comment",base)));

            admin.setExpand(simpleParse("expand",base));

            str=simpleParse("phrases",base);
            if(str!=null){
                strArray=str.split("\n");
                for(int i=0;i<strArray.length;i++){
                    strArray2=strArray[strArray.length-i-1].split(":");
                    admin.addPhrase(strArray2[0].trim(),strArray2[1].trim());
                }
            }
        } else {
            admin.setExpand(simpleParse("expand"));

            str=simpleParse("phrases");
            if(str!=null){
                strArray=str.split("\n");
                for(int i=0;i<strArray.length;i++){
                    strArray2=strArray[strArray.length-i-1].split(":");
                    admin.addPhrase(strArray2[0].trim(),strArray2[1].trim());
                }
            }
        }

        return admin;

    }

    public String simpleParse(String str){
        int i1=seq.indexOf(str);

        if(i1==-1)
            return null;
        int i2=seq.indexOf(";\n",i1);
            return seq.substring(i1+str.length(),i2).trim();
    }
    public String simpleParse(String str,String base){
        int i1=base.indexOf(str);

        if(i1==-1)
            return null;
        int i2=base.indexOf(";\n",i1);
            return base.substring(i1+str.length(),i2).trim();
    }
    public String simpleParse(String str,String base,String delimiter){
        int i1=base.indexOf(str);

        if(i1==-1)
            return null;

        if(delimiter==null)
            return base.substring(i1+str.length()).trim();
```

```
        else{
            int i2=base.indexOf(delimiter,i1);
            if(i2<0)
                return null;
            else
                return base.substring(i1+str.length(),i2).trim();
        }
    }
    public int simpleParseIndex(String str,String base,String delimiter){
        int i1=base.indexOf(str);

        if(i1==-1)
            return -1;
        int i2=base.indexOf(delimiter,i1);
        return i2;
    }

    public String toString(){
        return seq;

    }
}
//RCS projects properties
/*********************************************/
package common;

import java.util.*;
import java.io.*;

public class RCSProject implements Serializable{
    public String name;
    public Date date;
    public String desc;
    public String creator;
    public String fullPath;

    public RCSProject(String name){
        this.name=name;

    }
}
/*********************************************/
package client;

import java.awt.*;

public class Row implements Comparable{
          public int index;
    public boolean asc;
    private MyTableModel tableModel;
    public Row(MyTableModel tableModel){
        this.tableModel=tableModel;


    }
    public int compareTo(Object other){
              Row otherRow = (Row)other;
        Object a = tableModel.valueAt(index, tableModel.colNo);
            Object b = tableModel.valueAt(otherRow.index, tableModel.colNo);
            if(a==null){
            if(asc)
                return (b==null)?0:-1;
            else
```

```
                return (b==null)?0:1;
        } else if (a instanceof Comparable){

            if(asc){
                if(b==null)
                    return 1;
                return ((Comparable)a).compareTo(b);
            } else {
                if(b==null)
                    return -1;
                return -((Comparable)a).compareTo(b);
            }
             } else {
                    if(asc)
                return a.toString().compareTo((b==null)?"":b.toString());
            else
                return -a.toString().compareTo((b==null)?"":b.toString());
        }

    }
}
/*************************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.io.*;

public class SetBranchDialog extends JDialog{
    private Font font;
    private int option=JOptionPane.CLOSED_OPTION;

    private JComboBox c1;



    private String version;

    public SetBranchDialog(JFrame f,Font font,String fileName,String[]
versions){
        this(f,fileName,versions);
        this.font=font;


    }
    public SetBranchDialog(JFrame f,String fileName,String[] versions){
        super(f,"Set Default Branch",true);


        Container c=getContentPane();
        c.setLayout(new BorderLayout());

        JPanel p1=new JPanel();
        p1.setLayout(new GridBagLayout());
        p1.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
        c.add(p1,BorderLayout.CENTER);

        GridBagConstraints gc=new GridBagConstraints();
        gc.insets=new Insets(2,2,2,2);
        gc.anchor=GridBagConstraints.WEST;
```

```
JLabel l1=new JLabel("File Name:");
gc.gridwidth=1;
l1.setFont(font);
p1.add(l1,gc);



JLabel l2=new JLabel(fileName);
l2.setFont(font);
gc.gridx=1;
gc.gridwidth=3;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l2,gc);

JButton b1=new JButton("OK");
b1.setFont(font);
b1.setBorder(new BevelBorder(0));
gc.gridx=4;
gc.gridwidth=1;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(b1,gc);

JLabel l3=new JLabel("Branch:");
l3.setFont(font);
gc.gridx=0;
gc.gridy=1;
gc.gridwidth=1;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l3,gc);

c1=new JComboBox(versions);
c1.setFont(font);
c1.setBackground(Color.white);
c1.setMaximumRowCount(5);
c1.setSelectedIndex(versions.length-1);
version=versions[versions.length-1];
c1.addItemListener(
    new ItemListener(){
        public void itemStateChanged(ItemEvent e){
            setVersion();
        }
    }
);
gc.gridx=1;
gc.gridwidth=3;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(c1,gc);

JButton b2=new JButton("Cancel");
b2.setFont(font);
b2.setBorder(new BevelBorder(0));
gc.gridx=4;
gc.gridwidth=1;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(b2,gc);



b1.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
```

```
                        option=JOptionPane.OK_OPTION;
                        setVisible(false);


                    }
                }
            );

            b2.addActionListener(
                new ActionListener(){
                    public void actionPerformed(ActionEvent e){
                        option=JOptionPane.CANCEL_OPTION;
                        setVisible(false);
                    }
                }
            );
            setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
(Toolkit.getDefaultToolkit().getScreenSize().height-120)/2,400,120);
            show();
        }

        public void setVersion(){
            version=(String) c1.getSelectedItem();

        }
        public int getOption(){
            return option;
        }
        public String getBranchVersion(){
            return version;
        }


}
/*********************************************/
package client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.*;
import java.io.*;

public class SetCheckoutFolderDialog extends JDialog{
    private Font font;
    private int option=JOptionPane.CLOSED_OPTION;
    private JTextField t1=new JTextField();

    private String checkoutFolder;

    public SetCheckoutFolderDialog(JFrame f,Font font,String workingFolder){
        this(f,workingFolder);
        this.font=font;


    }
    public SetCheckoutFolderDialog(JFrame f,String workingFolder){
        super(f,"Set Checkout Directory",true);

        this.checkoutFolder=workingFolder;
        Container c=getContentPane();
        c.setLayout(new BorderLayout());
```

```
JPanel p1=new JPanel();
p1.setLayout(new GridBagLayout());
p1.setBorder(new EmptyBorder(new Insets(5,5,5,5)));
c.add(p1,BorderLayout.CENTER);

GridBagConstraints gc=new GridBagConstraints();
gc.insets=new Insets(2,2,2,2);
gc.anchor=GridBagConstraints.WEST;

JLabel l1=new JLabel("Folder Name:");
gc.gridwidth=1;
l1.setFont(font);
p1.add(l1,gc);


t1.setMinimumSize(new Dimension(320,20));
t1.setText(workingFolder);
gc.gridx=1;
gc.gridwidth=6;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(t1,gc);

JButton b1=new JButton("Browse...");
b1.setFont(font);
b1.setBorder(new BevelBorder(0));
gc.gridx=7;
gc.gridwidth=1;
p1.add(b1,gc);


JLabel l3=new JLabel("   ");
gc.gridx=0;
gc.gridy=1;
gc.gridwidth=2;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l3,gc);


JPanel p2=new JPanel();
gc.gridx=2;
gc.gridwidth=3;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(p2,gc);

JLabel l4=new JLabel("   ");
gc.gridx=5;
gc.gridwidth=3;
gc.weightx=1.0;
gc.fill=GridBagConstraints.HORIZONTAL;
p1.add(l4,gc);


p2.setLayout(new GridLayout(1,3,5,5));
JButton b2=new JButton("OK");
b2.setFont(font);
b2.setBorder(new BevelBorder(0));
p2.add(b2);

p2.add(new JLabel("  "));
```

```
        JButton b3=new JButton("Cancel");
        b3.setFont(font);
        b3.setBorder(new BevelBorder(0));
        p2.add(b3);


        b1.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    setCheckoutFolder();
                }
            }
        );


        b2.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                       if(!createDir()){
                        option=JOptionPane.OK_OPTION;
                        setVisible(false);
                    }

                }
            }
        );

        b3.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    option=JOptionPane.CANCEL_OPTION;
                    setVisible(false);
                }
            }
        );
        setBounds((Toolkit.getDefaultToolkit().getScreenSize().width-400)/2,
(Toolkit.getDefaultToolkit().getScreenSize().height-120)/2,400,120);
        show();
    }
    public boolean createDir(){
        boolean newDir=false;
        try{
            File f=new File(t1.getText().trim());
            if(!f.exists()){
                int answer=JOptionPane.showConfirmDialog(null,
                    "Folder doesn't exist. Create this folder?","Set Working
Folder",
                    JOptionPane.YES_NO_OPTION);
                if(answer==JOptionPane.OK_OPTION)
                    f.mkdir();

                newDir=true;
            }
            checkoutFolder=f.getAbsolutePath();
            if(!checkoutFolder.endsWith(Character.toString(File.separatorChar)))
                checkoutFolder+=File.separatorChar;

            t1.setText(checkoutFolder);

        } catch(Exception e) {};
        return newDir;
    }
    public void setCheckoutFolder(){
```

```
        JFileChooser chooser=new JFileChooser();
        chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int answer=chooser.showOpenDialog(this);
        if(answer==JFileChooser.APPROVE_OPTION){
            File f=chooser.getSelectedFile();
            checkoutFolder=f.getAbsolutePath()+File.separatorChar;
            t1.setText(checkoutFolder);
        }
    }
    public int getOption(){
        return option;
    }
    public String getCheckoutFolder(){
        return checkoutFolder;
    }


}
/*********************************************/
package toolkit;

import java.awt.*;
import java.io.*;
import java.util.*;
import java.util.regex.*;
import java.text.SimpleDateFormat;

public class Tools{
    public static SimpleDateFormat d2k=new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
    public Tools(){};
    public static Object[] fileToArray(String fileName){
        return fileToArray(new File(fileName));
    }
    public static Object[] fileToArray(File file){
            LinkedList l = new LinkedList();
            String s;
        int c;
        int last=-1;
            try{
            BufferedReader reader = new BufferedReader(new FileReader(file));
                    while ((s = reader.readLine()) != null)
                            l.add(s);
            reader.close();
            } catch (Exception e){ };

        try{
            BufferedReader reader = new BufferedReader(new FileReader(file));
            while((c=reader.read())!=-1)
                last=c;
            if(last==10)
                l.add(new String());
            reader.close();
        } catch (Exception e){ };

            return (String[]) l.toArray(new String[l.size()]);
        }
    public static Object[] stringToArray(String str){
        if(str==null)
            return null;
        LinkedList l = new LinkedList();
            String s;
            try{
```

```
            BufferedReader reader = new BufferedReader(new StringReader(str));
                    while ((s = reader.readLine()) != null)
                            l.add(s);
            reader.close();
            } catch (Exception e){ };
        if(str.endsWith("\n"))
            l.add(new String());
            return (String[]) l.toArray(new String[l.size()]);
    }
        public static String arrayToString(Object[] obj){
        return arrayToString(obj,"\n");
       }
    public static String arrayToString(Object[] obj,String separator){
        if(obj==null)
            return null;
            if(obj.length==0)
            return new String();
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < obj.length - 1; i++){
                    buf.append(obj[i]);
                    buf.append(separator);
            }
            buf.append(obj[obj.length - 1]);
            return buf.toString();
        }

    public static String RCSString(String str){
        if(str==null)
            return "@@";
        Pattern pattern=Pattern.compile("@");
        Matcher m=pattern.matcher(str);
        return "@"+m.replaceAll("@@")+"@";

    }
    public static String RCSString(Object[] obj){
        if(obj!=null)
            return RCSString(arrayToString(obj));
        else
            return "@@";
    }
    public static String nonRCSString(String rcsString){
        if(rcsString.length()<4)
            return null;
        Pattern pattern=Pattern.compile("@@");
        Matcher m=pattern.matcher(rcsString);
        String str=m.replaceAll("@");
        return str.substring(1,str.length()-1);
    }
    public static Date parseDate(String dateString){
        String[] ds=dateString.trim().split("\\.");
        int[] d=new int[6];
        for(int i=0;i<ds.length;i++)
            d[i]=Integer.parseInt(ds[i]);
        Calendar cal=new GregorianCalendar(d[0],d[1]-1,d[2],d[3],d[4],d[5]);
        Date date=cal.getTime();
        return date;
    }
}
/**********************************************/
package common;

import java.awt.*;
import java.io.*;
```

```
public class TreeNodeObject implements Serializable{
    public String name;
    public boolean isDir;
    public String fullPath;

    public String toString(){
        return name;
    }

}
/*
*This /*********************************************/
package defines RCS format and parser.
*/
/*********************************************/
package vcs;

import java.awt.*;
import java.util.*;
import java.util.regex.*;

/**
* A version is a serial of dot-separated digit numbers.
*
* @author <a href="mailto:zhuwang@yahoo.com">Zhu Wang</a>
* @version 1.0
*/
public class Version extends Object{
    private int[] verNumbers;

    public Version(){
        this(1,0);
    }
    public Version(int major){
        this(major,0);
    }

    public Version(Version ver){
        this.verNumbers=(int[]) ver.verNumbers.clone();
    }
    public Version(String ver){
        Pattern pattern=Pattern.compile("[\\d+\\.]*\\d+");
        Matcher m=pattern.matcher(ver);
        if(m.find()){
            if((m.end()-m.start())==ver.length()){
                String[] str=ver.split("\\.");

                if((str.length % 2)==0)
                    verNumbers=new int[str.length];
                else
                    verNumbers=new int[str.length+1];
                for(int i=0;i<str.length;i++)
                    verNumbers[i]=Integer.parseInt(str[i]);
                if(verNumbers.length>str.length)
                    verNumbers[verNumbers.length-1]=1;
            }

        }
    }
    public Version(int[] num){
        if(num.length % 2==0)
            verNumbers=(int[]) num.clone();
```

```
    else{
        verNumbers=new int[num.length+1];
        for(int i=0;i<num.length;i++)
            verNumbers[i]=num[i];
        verNumbers[verNumbers.length-1]=1;
    }


}
public Version(int major,int minor){
    verNumbers=new int[2];
    verNumbers[0]=major;
    verNumbers[1]=minor;
}
public boolean isBranch(){
    return (verNumbers.length>2);
}
public boolean isTrunk(){
    return (verNumbers.length==2);
}
public int compare(Version ver){
    if(this==ver)
        return 0;
    else{
        int size=Math.min(verNumbers.length,ver.verNumbers.length);
        for(int i=0;i<size;i++){
            if(verNumbers[i]>ver.verNumbers[i])
                return 1;
            else if(verNumbers[i]<ver.verNumbers[i])
                return -1;
        }
        if(verNumbers.length>ver.verNumbers.length)
            return 1;
        else if(verNumbers.length<ver.verNumbers.length)
            return -1;
        else
            return 0;

    }
}
public boolean isLarger(Version ver){
    return compare(ver)==1;
}
public int size(){
    return verNumbers.length;
}
public int nthVersion(int n){
    if(n>(size()-1) || n<0)
        return -1;
    return verNumbers[n];

}
public void setNthVersion(int n,int ver){
    if(n>=0 && n<size() && ver>=0)
        verNumbers[n]=ver;

}

public Version getBranchPoint(){
    int[] num=new int[verNumbers.length-2];
    for(int i=0;i<num.length;i++)
        num[i]=verNumbers[i];
    return new Version(num);
```

```
    }
    public Version getTrunkVersion(){
        int[] num=new int[2];
        for(int i=0;i<num.length;i++)
            num[i]=verNumbers[i];
        return new Version(num);
    }
    public Version nextVersion(){
        Version ver=new Version(this);
        ver.verNumbers[ver.verNumbers.length-1]++;
        return ver;
    }
    public Version addBranch(){
        int[] num=new int[verNumbers.length+2];
        for(int i=0;i<verNumbers.length;i++)
            num[i]=verNumbers[i];
        num[num.length-2]=1;
        num[num.length-1]=1;
        return new Version(num);

    }
    public String toString(){
        String str="";
        if(verNumbers!=null){
            for(int i=0;i<verNumbers.length-1;i++)
                str+=Integer.toString(verNumbers[i])+".";
            str+=Integer.toString(verNumbers[verNumbers.length-1]);
        }
        return str;
    }
}
```